UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UMA METODOLOGIA DE GERAÇÃO DE SEQÜÊNCIAS DE TESTE A PARTIR DE ESPECIFICAÇÕES DESCRITAS NA TÉCNICA DE DESCRIÇÃO FORMAL ESTELLE

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do grau de

MESTRE EM ENGENHARIA ELÉTRICA

Luis Otávio Rodrigues da Silva

Florianópolis, Agosto de 1994

UMA METODOLOGIA DE GERAÇÃO DE SEQÜÊNCIAS DE TESTE A PARTIR DE ESPECIFICAÇÕES DESCRITAS NA TÉCNICA DE DESCRIÇÃO FORMAL ESTELLE

Luis Otávio Rodrigues da Silva

Esta dissertação foi julgada para a obtenção do título de
Mestre em Engenharia
especialidade **Engenharia Elétrica,**
área de concentração **Controle, Automação e Informática Industrial,**
e aprovada em sua forma final pelo Curso de Pós-Graduação

Prof. Vitório Bruno Mazzola, Dr.
Orientador

Prof. Roberto de Souza Salgado, Ph.D.
Coordenador do Curso de Pós-Graduação em Engenharia Elétrica

Banca Examinadora :

Prof. Vitório Bruno Mazzola, Dr.(Presidente)
Orientador

Prof. Jean Marie Farines, Dr. Ing.

Prof. Joni da Silva Fraga, Dr.

Prof. Akebo Yamakami, Dr.

À minha família pelo apoio e carinho dado nas horas difíceis e à Lorena pela paciência e boa vontade com que me esperou durante todo esse tempo.

AGRADECIMENTOS

Meus sinceros agradecimentos ao Prof. Vitório Bruno Mazzola não só pela orientação na realização deste trabalho, mas também pela sua amizade durante este tempo, demonstrando que antes de ser um ótimo orientador é um grande amigo.

Agradecimentos aos membros da banca examinadora pela aceitação de participação e pelas críticas e comentários.

Agradecimentos a todos os professores do curso de pós-graduação que contribuíram com o meu enriquecimento profissional e aos colegas e funcionários do departamento que, direta ou indiretamente, contribuíram na realização deste trabalho.

Aos meus colegas de república Alexandre (exótico), Idmilson (Tomé-Açu), Aberides (Pelé) e Junior (Apuca) pela convivência e por terem me aturado.

Aos amigos Celito, Ana Luíza e família, Ronchi, Letícia e família pelo apoio dado aqui em Florianópolis.

Aos meus colegas de turma Alberto, Itsumi, Salomão, Tereza, Carla, André, João Manoel e aos colegas de laboratório Leonardo Kammer e Paulo Valim pela troca de idéias.

À galera de base : Maurício (Osmany), Marcos (gumex), Márcio Suguieda (Norton Jr), Jáder (Indiana Jones), Cícero (Cicinho), Mussoi (petequinha), Thair (turco), Busch (alemão), Aldebaro (gazela), Polyana, Gisele, Juan (nanico), Gláucio (baiúcho), Ricardo (macaco), Eduardo (dudinha), André (maluco), Paulo (The Flash), Daniel (negão), Luiz Henrique (Zique) pela convivência aqui em Florianópolis.

Um agradecimento à UFSC e ao programa PICD-CAPES da Universidade Federal do Pará pelo suporte material e financeiro.

Agradeçimento especial à minha mãe por não ter me deixado desistir no início do curso.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIGLAS E NOTAÇÕES

ASP	Abstract Service Primitives
ATM	Abstract Test Method
ATS	Abstract Test Suite
CCITT	International Consultative Committee for Telephones and Telegraphs
EFSM	Extended Finite State Machine
FDT	Formal Description Techinique
FSM	Finite State Machines
I/O	Input/Output
ISO	International Organization for Standardization
ITU	International Telecomunication Union
IUT	Implementation Under Test
LAAS-CNRS	Laboratoire d'Automatique et d'Analyse des Systèmes du Centre National de la Recherche Scientifique
LCMI	Laboratório de Controle e Microinformática
LT	Lower Tester
OSI	Open System Interconnection
PCO	Point of Control and Observation
PCTR	Protocol Conformance Test Report
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Information
PIXIT	Protocol Implementation eXtra InformaTion
RPC	Remote Procedure Call

SAP	Service Acess Point
SCTR	System Conformance Test Report
SUT	System Under Test
TCP	Test Coordination Procedures
TOF	Test Oriented Form
TTCN	Tree and Tabular Combined Notation
UFSC	Universidade Federal de Santa Catarina
UT	Upper Tester

RESUMO

O trabalho aqui apresentado enquadra-se na área da Engenharia de Protocolos de Comunicação, mais precisamente no domínio do Teste de Conformidade de Protocolos. O objetivo visado neste foi, num primeiro tempo, a aquisição de conhecimentos neste domínio e, num segundo tempo, aportar uma contribuição através da definição de uma metodologia de geração de sequências de teste a partir de especificações formais.

A partir do estudo das diversas técnicas de especificação formal existentes, adotou-se a técnica de descrição formal ESTELLE, padronizada na ISO, como objeto de estudo neste trabalho.

A fim de atingir o primeiro objetivo fixado, foi realizado um estudo bastante aprofundado dos diversos aspectos relacionados ao Teste de Conformidade de Protocolos de Comunicação, principalmente os seguintes: a padronização dos Testes de Conformidade junto a ISO, as metodologias de aplicação do Teste de Conformidade, a técnica de representação de testes (TTCN) padronizada na ISO, as técnicas de geração de sequências de teste.

No que diz respeito a este último ponto, e visando o segundo objetivo fixado, foi investigada a possibilidade de definição de uma metodologia para a geração de sequências de teste de conformidade. Esta metodologia é baseada em dois dos trabalhos encontrados na literatura estudada. O primeiro, definindo uma técnica de geração de sequências de teste para especificações de protocolos de comunicação descritas em Máquinas de Estados Finitos, o método UIO (*Unique Input Output*). O segundo, é baseado nos resultados obtidos no LAAS-CNRS (França) no que diz respeito à definição de uma versão estendida de ESTELLE, denominada ESTELLE*, e no desenvolvimento de uma ferramenta (ESTIM), que permite validar especificações descritas nesta versão de ESTELLE.

O resultado da fusão destes dois estudos foi a metodologia proposta neste trabalho. Esta metodologia consiste na obtenção, a partir de uma especificação formal descrita em ESTELLE*, de um autômato (máquina de estados) representando o comportamento observável da entidade de protocolo a ser testada. Em seguida é aplicado o método UIO sobre o modelo máquina de estados.

Esta metodologia é concretizada a partir de um conjunto de ferramentas implementadas em linguagem C, num ambiente de estações de trabalho SUN SPARC, operando sob UNIX.

Finalmente, a metodologia proposta é ilustrada pela aplicação a exemplos extraídos da literatura estudada.

ABSTRACT

This work is related to the Protocol Engineering area, more precisely on the domain of Conformance Testing. The goals established here were twofold: first, a global study of the main research results on this domain, and, following, contribute to these results by defining a methodology for generating test suites from protocol formal specifications.

From the analysis of several formal techniques, the ESTELLE formal description technique, standardized by ISO, has been selected in this work.

In order to reach the first goal in this work, a study of the main results in research on Conformance Testing has been done, considering several points: standardization at ISO, methodologies of Conformance Testing, a technique for describing testing (TTCN) standardized at ISO, techniques for generating test suites from formal specifications.

Related to this last point and aiming at the second goal established, the possibility of defining a methodology for generating test suites from formal specifications was investigated. This methodology is based on two main results studied in the literature. The first one, defining a technique of test sequence generation for protocols specifications described in finite state machines, the method UIO (*Unique Input Output*). The second one is based on the results obtained at LAAS-CNRS, related to the definition of a ESTELLE extended version named ESTELLE* and in the development of a tool, ESTIM, conceived for validating specifications described in this version of ESTELLE.

The result of the fusion of these two studies was the methodology proposed in this work. This methodology consist in obtaining, from formal specifications described in ESTELLE*, an automaton (state machine) representing the observable behaviour of the protocol entity to be tested. Afterwards, UIO method is applied on the machine state model.

This methodology is achieved from a set of tools implemented in C on the SUN SPARC environment, operating under UNIX.

Finally, the methodology proposed is illustrated by the application on examples extracted from the literature.

CAPÍTULO 1

INTRODUÇÃO

O projeto de sistemas distribuídos abertos tem recebido bastante atenção nos últimos 15 anos. A definição do modelo de referência OSI[1], **[Zimmermann 80]** e a normalização de um extenso conjunto de protocolos de comunicação foram, sem dúvida, grandes passos dados em direção ao desenvolvimento das redes de computadores.

Os sistemas de protocolos de comunicação não são iguais aos sistemas de *software* tradicionais, e por este motivo eles têm conceitos especiais para o projeto e a implementação. Os sistemas tradicionais consistem de funções que vão de um estado inicial para um estado final, aceitando todas as entradas no início de suas operações e proporcionando suas saídas no final, estes sistemas são chamados de *sistemas transformacionais*, por que eles transformam um estado inicial num estado final, com a velocidade fixada pelo próprio sistema **[Rayner 87]**.

Porém alguns sistemas como sistemas operacionais e sistemas para controle de processos podem nunca terminar, estes sistemas são chamados de *sistemas reativos*. Um sistema reativo é um sistema que mantém uma interação contínua com o ambiente, reagindo com os estímulos segundo uma velocidade fixada pelo ambiente externo.

Os sistemas de protocolos de comunicação são *sistemas reativos* com algumas características próprias. Cada entrada do protocolo pode não ter uma saída correspondente, e uma entrada pode ter muitas saídas. Uma saída correta de um protocolo depende dos valores de suas saídas precedentes.

A especificação de um protocolo é baseada no serviço de comunicação a ser proporcionado e o serviço de comunicação que o protocolo irá utilizar (serviço oferecido pela camada inferior ou pela rede). Os projetistas devem checar a especificação do protocolo para garantir que ele é correto e consistente, proporciona o serviço de comunicação desejado e oferece estes serviços com uma eficiência aceitável.

---

[1]OSI (Open Systems Interconnection)

Devido a esta grande complexidade dos protocolos de comunicação houve um grande esforço no desenvolvimento de trabalhos voltados à definição de modelos e metodologias para o seu desenvolvimento. Podemos destacar aí a aplicação de modelos baseados em sistemas de transição de estados, como as máquinas de estado finitos (FSM[2]), **[Danthine 80]**, as Redes de *Petri*, **[Diaz 82]**, e a definição de técnicas de descrição formal (FDT[3]), feitas no seio dos organismos internacionais de normalização. Exemplos destas técnicas são ESTELLE e LOTOS definidas pela ISO, **[ISO9074]**, **[ISO8807]** e SDL definida pelo ITU (*International Telecomunication Union*, antigo CCITT), **[CCITT/Z-100]**.

Estas técnicas têm desempenhado, nos últimos anos, um papel bastante importante no que diz respeito à concepção de protocolos de comunicação e mesmo de outras classes de aplicações distribuídas. Ferramentas de *software*, desenvolvidas em torno destas técnicas, permitem hoje automatizar as principais etapas de concepção de um protocolo ou de uma aplicação distribuída.

Uma etapa de grande importância no ciclo de vida de um protocolo de comunicação é o teste de conformidade. É através da realização desta etapa que uma implementação vai ser confrontada no que diz respeito à sua coerência com a norma que definiu o protocolo em questão.

1.1 - O Significado de Conformidade em OSI

Conformidade é uma das noções chaves no contexto de sistemas abertos. Sendo conforme a um padrão internacional um sistema se torna aberto para todos os outros sistemas que utilizam o mesmo padrão, contudo, a noção de conformidade deve ser precisamente definida.

Conformidade no contexto da OSI diz respeito apenas à conformidade das implementações e sistemas reais para os padrões OSI, os quais especificam restrições aplicáveis.

Existem 3 caminhos para se olhar as restrições de conformidade :

1. As restrições de conformidade num padrão podem ser :

   - *restrições obrigatórias* : devem ser observadas em todos os casos;

---

[2]FSM (Finite State Machines)
[3]FDT (Formals Description Techiniques)

- *restrições condicionais* : devem ser observadas se as condições ajustadas no padrão podem ser aplicadas;

- *restrições opçionais* : podem ser selecionadas para ajustar a implementação, proporcionando a observação de qualquer restrição aplicada à especificação.

2. As declarações das restrições de conformidade em um padrão podem ser iniciadas :

   - *positivamente* : iniciam com o que deve ser feito;

   - *negativamente* (proibições) : iniciam com o que não deve ser feito.

3. As restrições de conformidade se dividem em 2 grupos :

   - *restrições de conformidade estáticas*

   - *restrições de conformidade dinâmicas*

Deve ser feita uma clara distinção entre as *restrições de conformidade estáticas* e *dinâmicas*. Para evitar ambigüidade elas devem ser iniciadas separadamente uma da outra.

As *restrições de conformidade estáticas* são aquelas que definem as mínimas capacidades permitidas de uma implementação. Estas restrições podem estar em um nível mais baixo, tais como o agrupamento de unidades funcionais e opções em classes de protocolos, ou podem estar em um nível mais detalhado, tais como o alcance dos valores que devem ser suportados para parâmetros específicos e temporizadores.

Se as *restrições de conformidade estáticas* não estão completamente especificadas então, todas as capacidades iniciadas implicitamente como restrições de conformidade estáticas têm que ser consideradas opcionais.

As *restrições de conformidade dinâmicas* são restrições e opções que definem um conjunto de comportamentos permitidos de uma implementação em exemplos de comunicação. Este conjunto define as capacidades máximas que uma implementação conforme pode ter dentro dos termos do padrão de protocolo OSI. Então, *restrições de conformidade dinâmicas* são todas aquelas restrições que determinam qual comportamento observável é permitido pelos padrões de protocolo OSI em exemplos de comunicação.

Um sistema ou implementação conforme deve satisfazer as *restrições estáticas* e as *dinâmicas* e ser consistente com as capacidades e opções iniciadas no PICS[4]. Sendo que o PICS consiste de uma declaração, feita pelos fornecedores de uma implementação ou sistema OSI, iniciando as capacidades e opções que devem ser implementadas e qualquer aspecto que deve ser omitido.

1.2 - A Importância dos Testes de Conformidade

Uma etapa de grande importância no ciclo de vida de um protocolo de comunicação é o teste de conformidade. É através da realização desta etapa que uma implementação vai ser confrontada no que diz respeito à sua coerência com a norma que definiu o protocolo em questão.

A realização do teste de conformidade é baseada na aplicação, à implementação, de um conjunto de dados de entrada, ou simplesmente *entradas* e a verificação, com base no conjunto de dados obtidos na saída, ou *saídas*, se a implementação apresenta ou não o mesmo comportamento que o da especificação. Se isto for verdade, então diz-se que a implementação está *conforme* com a especificação. A conformidade é definida em dois níveis, *fraca* e *forte* :

- *Conformidade Forte* : uma implementação tem conformidade forte para uma especificação se ambas geram as mesmas saídas para todas as seqüências de entrada, ou seja, os comportamentos desejável e indesejável da implementação são os mesmos que os da especificação da FSM.

- *Conformidade Fraca* : uma implementação tem conformidade fraca se ela possui o mesmo comportamento desejável que o da especificação da FSM, porém o comportamento indesejável pode não ser igual ao da especificação da FSM.

A etapa de teste de conformidade, pela sua importância, deve ser caracterizada por dois parâmetros, de certo modo, antagônicos: a *eficiência*, que sugere um menor dispêndio de tempo possível na realização desta etapa; e a *eficácia*, que está relacionada ao grau de cobertura ou abrangência da atividade de teste.

Assim sendo, o teste de conformidade da implementação de um protocolo deve ser *eficiente* e *eficaz*. É evidente que, dado o antagonismo destas duas definições, a satisfação de

---

[4]PICS (Protocol Implementation Conformance Information)

uma delas pode resultar no desrespeito total à outra definição. A solução, neste caso, é encontrar o maior equilíbrio possível entre estes dois parâmetros, o que, evidentemente, não é uma tarefa das mais simples.

Uma coisa, no entanto é clara. A obtenção de um equilíbrio satisfatório entre *eficiência* e *eficácia* está fortemente ligada à metodologia empregada para a obtenção do conjunto de dados a ser aplicado à implementação. A esta atividade é dado o nome de *geração de seqüências de teste* e, para isto, o uso de métodos de geração de seqüências de teste baseados em técnicas formais pode ser a chave para a obtenção de um conjunto de seqüências que determinem o equilíbrio entre *eficiência* e *eficácia*.

1.3 - Divisão do Trabalho

O texto do trabalho foi elaborado de modo a fornecer uma visão geral de todo o processo de teste de protocolos de comunicação. No capítulo 2 é mostrada de uma forma suscinta todas as atividades do processo de teste, a estrutura básica de um pacote de teste, os tipos de testes, os métodos de testes propostos pela ISO e a notação TTCN.

O capítulo 3 diz respeito aos problemas associados com a testabilidade, neste capítulo são tratadas as questões do não-determinismo e da coordenação suas implicações no teste e são discutidas técnicas para se obter melhoras na testabilidade. Ainda neste capítulo é mostrado o uso de técnicas de descrição formal na implementação, o poder de expressão das linguagens de espeçificação e linguagens de implementação, a relação entre o estilo de especificação e implementação e as escolhas na implementação.

O capítulo 4 é reservado à geração de seqüências de teste, neste capítulo são apresentados métodos de geração baseados em FSM's com exemplo de aplicação baseado no modelo de FSM apresentado. Além disso é feita uma comparação entre estes métodos apresentando as vantagens e desvantagens de cada um. São apresentados, ainda neste capítulo, metodologias definidas para especificações utilizando técnicas de descrição formal, como ESTELLE e LOTOS.

O capítulo 5 diz respeito a apresentação de uma metodologia para a geração de seqüências de teste a partir da técnica de descrição formal ESTELLE utilizando um método baseado em FSM's. Neste capítulo é apresentada a ferramenta que implementa esse método bem como alguns exemplos de aplicação para esta ferramenta.

As conclusões e perspectivas deste trabalho são devidamente encontradas no capítulo 6. Após este capítulo estão disponíveis o conjunto de referências bibliográficas e alguns anexos que serão descritos no decorrer do texto.

CAPÍTULO 2

O TESTE DE PROTOCOLOS DE COMUNICAÇÃO

2.1 - Introdução

Como já foi mencionado o teste de protocolos de comunicação é uma etapa de grande importância para o ciclo de vida de um protocolo. Com isso, grupos internacionais de padronização, tais como a ISO (*International Organization for Standardization*) e o ITU (*International Telecomunication Union*), têm definido os conceitos básicos relacionados com os testes de conformidade de implementações de protocolos. Os esforços de padronização têm sido direcionados principalmente às questões que consideram a implementação de pacotes de testes em laboratórios de teste, estes esforços resultaram num padrão dividido em 5 partes. Os métodos para geração dos testes de conformidade não estão incluídos neste esforço por causa da necessidade de um método formal que esteja de acordo com todas as principais organizações de teste.

Os assuntos que são incluídos no padrão de 5 partes são os seguintes : conceitos gerais dos testes de conformidade (parte 1), especificação de pacotes de teste (parte 2), a notação TTCN (*Tree and Tabular Combined Notation*) (parte 3), realização de testes (parte 4) e restrições nos laboratórios de testes e clientes para o processo de avaliação de conformidade (parte 5).

A metodologia de testes apresentada pela ISO é baseada no modelo de referência OSI de sete camadas, onde cada entidade de camada deste modelo, chamada entidade (N), fornece certos serviços, chamados serviços (N), para a sua entidade superior. Para proporcionar estes serviços à entidade (N+1), uma entidade (N) faz uso dos serviços fornecidos pela sua entidade inferior (isto é, a entidade (N-1)). Uma entidade (N) junto com os serviços que ela proporciona à entidade (N+1) é chamada de fornecedor do serviço (N).

Duas entidades (N) pares usam (N)-PDU's (*Protocol Data Unit*) para se comunicar através de um fornecedor de serviço (N-1). Um protocolo (N) é um conjunto de regras e convenções que definem a comunicação entre duas entidades pares.

As (N)-PDU's definidas para um protocolo (N) podem ser classificadas em 3 grupos : PDU's *válidas*, *inoportunas* e *ilegais*. As PDU's *válidas* são definidas para o funcionamento

normal (ou esperado) de uma entidade (N). As PDU's *inoportunas* (PDU's que são sintaticamente e semanticamente corretas, mas que chegaram inesperadamente) representam o comportamento *inesperado* da entidade (N) remetente. As PDU's *ilegais* não encontram as restrições de sintaxe do protocolo (N), correspondendo ao comportamento *errado* da entidade (N) remetente. Uma especificação de um protocolo (N) é *completa* se as ações tomadas pela entidade (N) são claramente definidas após o recebimento de qualquer um dos 3 tipos de PDU's.

Uma entidade (N) se comunica com sua entidade superior e inferior através dos ASP's (*Abstract Service Primitives*) (N)- e (N-1)- respectivamente. Numa implementação de uma entidade (N), os ASP's são implementados nos SAP's (*Service Acess Point*). Os PCO's (*Point of Control and Observation*), pontos onde uma implementação pode ser controlada e observada, de uma implementação definem o comportamento externamente controlável e observável de uma camada (N). Durante os testes de conformidade, os ASP's e PDU's que são definidos, são enviados à implementação da entidade (N) e os ASP's e PDU's gerados são observados.

Como foi visto no capítulo anterior existem dois níveis de conformidade, *fraca* e *forte*, então os testes para conformidade forte deve verificar se a implementação manuseia corretamente todos os tipos de PDU's (*válidas*, *inoportunas* e *ilegais*)

A divisão deste capítulo traz, na seção seguinte, o processo de teste definido pela ISO. Na seção 2.3 apresenta-se a estrutura de um pacote de teste; na seção 2.4 são discorridos os diversos tipos de teste aplicados aos protocolos de comunicação; nas seções 2.5 e 2.6 são mostrados, respectivamente, os métodos de testes de protocolos e a notação TTCN definidos pela ISO. E, por fim, na seção 2.7 são apresentadas as conclusões específicas do capítulo.

2.2 - O Processo de Teste

A ISO dividiu o processo de teste de uma IUT[5] em alguns passos mostrados abaixo, porém esta descrição é idealizada, visto que alguns passos podem ser combinados, omitidos e/ou reordenados :

[5]Uma implementação que está sendo testada é chamada de IUT (Implementation Under Test).

- desenvolver no mínimo um pacote de teste abstrato padronizado, usando uma das metodologias de teste;

- desenvolver uma declaração de um PICS (*Protocol Implementation Conformance Statement*). O PICS consiste das declarações de capacidades e opções que têm sido implementadas para a especificação do protocolo base para qual a conformidade é exigida;

- desenvolver uma declaração de um PIXIT (*Protocol Implementation eXtra InformaTion*). O PIXIT fornece informações específicas da implementação, relativamente a IUT que é necessária para teste (por exemplo, valores exigidos para os temporizadores, elementos endereçáveis numa pilha de uma entidade de protocolo).

A metodologia da ISO identifica os dois últimos passos como partes exigidas de um padrão de protocolo. Cada grupo de definição do protocolo OSI é responsável pela especificação das exigências de conformidade associadas com o protocolo. Eles também são responsáveis pela garantia de que um PICS é produzido consistente com essas exigências de conformidade, como uma parte separada ou um anexo da especificação do protocolo.

O PICS deve cobrir todas as funções condicionais e opcionais, elementos de procedimentos, parâmetros, opções, PDU's, temporizadores, e outras capacidades identificadas na especificação do protocolo.

Um PICS não repete a especificação do protocolo. Ele captura a flexibilidade da implementação permitida por essa especificação. Além disso, ele detalha quais opções são deixadas para o implementador, e quais são condicionalmente dependentes de outras opções tomadas pelo implementador.

O PICS é usado por um laboratório de testes para garantir que aspectos obrigatórios são incluídos e os aspectos condicionais são também implementados como exigido. O PICS também é usado para seleção, isto é, o laboratório de teste elimina os testes para os aspectos e as opções que não foram implementados.

A informação contida num PIXIT é usada para parametrizar um pacote de teste, por exemplo, temporizadores e endereços de cada componente de protocolo de uma IUT podem ter que ser setados antes do teste começar.

As regras para a construção e definição formal do PICS e do PIXIT podem ser encontradas nas partes 1 e 2 do documento de padronização proposto pela ISO **[ISO9646-1]** e **[ISO9646-2]**.

Os testes são executados, resultados são analisados e dois relatórios são gerados. Um PCTR (*Protocol Conformance Test Report*) contém informações suficientes para identificar o sistema e o protocolo testado, identificar o pacote de teste e o método de teste empregado. Pode existir mais de um PCTR, visto que mais de um protocolo pode ser testado. Um SCTR (*System Conformance Test Report*) identifica um sistema o qual foi testado e fornece uma declaração de conformidade resumida para cada componente do protocolo testado.

2.2.1 - Atividades do Processo de Teste

O processo de teste envolve algumas atividades que são comuns a todos os tipos de testes de protocolos (testes de diagnóstico, conformidade, interoperabilidade, desempenho e robustez). Estas atividades incluem :

- geração do teste;
- verificação dos testes;
- seleção e parametrização dos testes.

2.2.1.1 - Geração do Teste

É o processo de derivação de casos de teste a partir de uma especificação do protocolo.

Fazendo hipóteses a respeito da especificação, muitos métodos existentes para a geração de seqüências de teste assumem que a especificação é dada como uma FSM determinística completamente especificada. Além disso, esses métodos em geral cobrem apenas aspectos de controle do protocolo. Outros aspectos importantes, tais como, o não-determinismo e aspectos de dados não são considerados.

Para se obter casos de testes efetivos, todos os aspectos do protocolo devem ser considerados durante a geração dos casos de teste. Apenas recentemente algum trabalho tem sido desenvolvido na geração de testes e diagnóstico de falhas para protocolos modelados como FSM não-determinísticas **[Arakawa 91]**, **[Fujiwara 91]** e **[Ghedamsi 92]**, enquanto que muitos resultados já estão disponíveis para os determinísticos **[Sidhu 89]**.

Um outro aspecto que influencia o processo de geração de casos de teste são as interfaces de teste. Alguns detalhes dos casos de teste devem mudar por causa das interfaces de

teste. Por exemplo, se alguma restrição de tempo necessitar ser testada, os casos de teste devem considerar o atraso introduzido pelo provedor do serviço. Além disso, uma especificação de protocolo deve especificar o comportamento da entidade de protocolo em ambos os pontos de acesso ao serviço superior e inferior, isto é, (N)-SAP's e (N-1)-SAP's. No nível de implementação dois ou mais SAP's podem corresponder a um SAP no nível de especificação. Contudo, quando um protocolo está sendo testado é necessário considerar o mapeamento de SAP's da implementação para a especificação.

Se um SAP no nível de especificação é representado por dois ou mais SAP's no nível de implementação deve-se ter um comportamento não-determinístico da IUT devido a não-observabilidade. Isto é devido ao fato que quando eventos ou ações são oferecidos em dois ou mais SAP's, pode não ser possível ordená-los e consequentemente predizer se o comportamento está correto ou não.

Se uma correspondência um a um entre os SAP's dos níveis de especificação e implementação fosse observada, o trabalho do teste em checar se o comportamento da implementação segue o da especificação seria facilitado.

O processo de geração dos casos de teste a partir de uma especificação formal também deve considerar o problema do manuseamento de eventos inesperados.

Uma possível solução é considerar dois tipos de eventos : *válidos* e *não-especificados* (eventos *inoportunos* e *inválidos*). Estes eventos não-especificados devem ser manuseados no nível de implementação de acordo com o modelo semântico do método formal a partir do qual a implementação é derivada. Isto deve levar a diferentes comportamentos, uma vez que os modelos semânticos dos métodos formais não são completamente idênticos.

Implementações do mesmo protocolo de comunicação, porém, derivadas de diferentes especificações formais do protocolo devem exibir comportamentos consistentes quando os mesmos eventos são oferecidos em estados similares. Isto irá permitir a diferentes implementações interoperar. Para se ter diferentes implementações que se comportam consistentemente é necessário exigir algum tipo de *"consistência de eventos"*. A idéia é checar e transformar o evento antes que ele seja oferecido ao módulo que irá manuseá-lo. Se o evento é inválido ele pode ser recolocado por um evento apropriado esperado pelo módulo e com isso os eventos inválidos seriam manuseados apropriadamente.

Os casos de teste que cobrem os eventos não-especificados podem ser gerados dependendo de algum critério ou restrição na especificação.

2.2.1.2 - Verificação dos Testes

É o processo de checar se o comportamento esperado, a partir de um caso de teste, quando aplicado a uma IUT, é de fato válido com a especificação.

A verificação é exigida para os casos de testes derivados manualmente. Se a especificação do protocolo é escrita usando um método formal e casos de teste são derivados a partir da especificação, então a verificação dos testes não é necessária.

Para evitar possíveis erros na determinação de veredictos para resultados de teste, os casos de teste devem ser gerados automaticamente a partir de uma especificação formal do protocolo.

2.2.1.3 - Seleção e Parametrização dos Testes

Uma implementação de protocolo não necessita suportar todas as funções e aspectos presentes na especificação. Portanto, apenas um subconjunto dos possíveis casos de teste podem ser aplicados para a implementação. O processo de identificação do subconjunto correto dos casos de teste é chamado seleção dos testes. A parametrização dos testes é o processo de determinação dos valores para os parâmetros dos casos de teste.

2.2.1.4 - Execução dos Testes

Para testar uma implementação, deve-se gerar casos de testes executáveis a partir de casos de testes selecionados e parametrizados.

O sistema de execução do teste deve permitir a comunicação entre um caso de teste e uma implementação, proporcionando uma maneira para monitorar a execução do teste e um mecanismo para registrar a história da execução de um caso de teste para uma análise adicional. Desta maneira deve-se traçar casos de testes nos quais ocorre uma falha na implementação quando comparada com a especificação formal do protocolo e diagnosticar as razões para esta falha.

2.3 - Estrutura de um Pacote de Teste

Os testes são representados numa estrutura hierárquica num pacote de teste de conformidade abstrato, esta estrutura está ilustrada na figura 2.1, sendo que os seus diversos elementos serão definidos abaixo :

Grupos de Teste : consistem de vários casos de teste de acordo com o ordenamento lógico de execução.

Casos de Teste : este é o nível chave, baseado num propósito de teste estritamente definido. Um propósito de teste descreve o objetivo do caso de teste tão especificadamente quanto possível.

Passos de Teste : um caso de teste pode conter passos de teste nomeados, cada um dos quais consistindo de vários eventos de teste.

Eventos de Teste : interações atômicas entre a IUT e os mecanismos de teste.

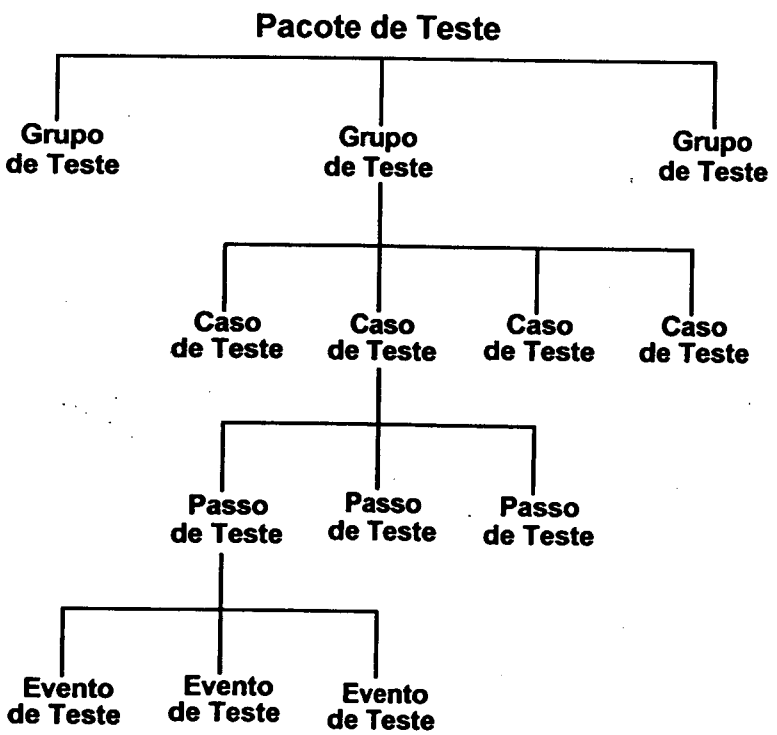

Figura 2.1. - Estrutura do Pacote de Teste

2.4 - Tipos de Teste

O objetivo principal dos testes de conformidade é verificar se a implementação que está sendo testada é conforme com a especificação do padrão. Limitações práticas tornam isso impossível de ser exaustivo e considerações econômicas podem restringir o teste ainda mais.

Todavia, existem 4 tipos de testes distintos, de acordo com o grau que eles proporcionam uma indicação de conformidade : os testes de interconexão básica, os testes de capacidade, os testes de comportamento e os testes de resolução de conformidade.

2.4.1 - Teste de Interconexão Básica

Proporcionam testes limitados de uma IUT em relação aos principais aspectos num padrão, permitindo verificar se existe conformidade suficiente para a interconexão ser possível, sem tentar executar através de testes.

Estes testes são apropriados para detectar casos severos de não-conformidade, servir como um filtro preliminar antes de se aplicar testes mais apurados; dar uma primeira indicação de que uma implementação, a qual passou por um teste de conformidade completo num ambiente, continua conforme num novo ambiente; determinar se uma implementação pode ser usada para comunicação com outras implementações conformes, por exemplo como uma troca de dados preliminar.

Estes testes devem ser padronizados tanto como um pacote de testes muito pequeno ou como um subconjunto de um pacote de testes de conformidade. Eles podem ser usados sozinhos ou junto com um pacote de testes de conformidade.

2.4.2 - Testes de Capacidade

Proporcionam testes limitados de cada uma das restrições estáticas de conformidade num padrão (são aquelas que definem as mínimas capacidades permitidas de uma implementação), para apurar qual capacidade de uma IUT pode ser observada e para checar quais dessas capacidades são válidas com respeito às restrições de conformidade estáticas e ao PICS.

São apropriados para checar no máximo possível a consistência de um PICS com uma IUT, checar que as capacidades de uma IUT são consistentes com as restrições de

conformidade estáticas, possibilitar uma seleção eficiente dos testes de comportamento a serem realizados para uma IUT particular e servir como uma base para assegurar conformidade quando realizados junto com os testes de comportamento. Eles devem ser padronizados dentro de um pacote de testes de conformidade. Eles podem tanto estar separados dentro de seus próprios grupos de teste ou fundidos com os testes de comportamento.

2.4.3 - Testes de Comportamento

Testam uma implementação tão completamente quanto é prático, sobre o alcance total das restrições dinâmicas de conformidade (exigências e opções que definem o conjunto do comportamentos permitidos de uma implementação ou de um sistema real em exemplos de comunicação, ou seja estas restrições definem as capacidades máximas que uma implementação conforme ou um sistema real pode ter dentro dos padrões de protocolos OSI) especificadas num padrão. Desde que o número de combinações possíveis dos eventos e o tempo dos eventos é infinito, tais testes não podem ser exaustivos. Uma limitação adicional é que este tipo de teste é projetado para ser executado coletivamente num único ambiente de teste, tal que qualquer falha que seja difícil de ser detectada neste ambiente provavelmente será perdida, por esta razão é possível que uma implementação não-conforme passe por um pacote de testes de conformidade. Eles devem ser padronizados como o corpo de um pacote de testes de conformidade.

2.4.4 - Testes de Resolução de Conformidade

Proporcionam respostas, tão próximas das definitivas quanto possível, para a questão de uma implementação satisfazer restrições particulares. Devido a problemas de exaustividade as respostas definitivas são atingidas às custas de testes de conformidade para um campo estreito.

A diferença entre os testes de comportamento e os testes de resolução de conformidade pode ser ilustrada pelo caso de um evento tal como o RESET. Os testes de comportamento devem incluir apenas uma seleção representativa das condições sobre as quais um RESET pode ocorrer e pode falhar para detectar um comportamento incorreto em outras circunstâncias. Os testes de resolução de conformidade seriam limitados às condições sobre as quais o comportamento incorreto já era suspeito de ocorrer, e confirmará ou não se a suspeita era correta. Esta classe de testes permite: obter respostas sim/não numa situação estritamente limitada e previamente identificada (por exemplo durante o desenvolvimento de uma implementação para checar se um aspecto particular foi implementado corretamente, ou durante o uso operacional, para investigar as causas dos problemas); identificar e oferecer resolução para

deficiências em pacote de testes de conformidade. Os testes de resolução de conformidade não são padronizados.

Em adição aos testes de conformidade, outros tipos de testes têm sido propostos, incluindo: *testes de interoperabilidade* (para determinar se duas IUT's irão de fato inter-operar e se não, porque não); *testes de desempenho* (para medir as características de desempenho de uma IUT); *testes de robustez* (para determinar o quanto bem uma IUT recupera-se de várias condições de erro), *testes de stress* (para examinar o comportamento de uma IUT sobre condições de carga máxima).

Pode existir alguma sobreposição entre estes tipos de testes e os testes de conformidade, se isto ocorrer então estes testes podem ser colocados como um subconjunto de um pacote de testes de conformidade completo.

2.5 - Métodos de Teste Propostos pela ISO

Antes de se começar a descrever os diversos métodos de teste, alguns conceitos básicos devem ser apresentados :

Abstract Test Suite (ATS). Um ATS constitui-se de um conjunto de casos de testes e opcionalmente passos de teste para um método de teste particular.

Abstract Test Method (ATM). Um ATM descreve uma arquitetura de teste abstrata consistindo de um *lower tester*, *upper tester* e procedimentos de coordenação de testes, e suas relações para o sistema de teste e o SUT[6]. Cada ATM determina os pontos de controle e observação (PCO's) e os eventos de teste (isto é, as primitivas de serviços abstratos e as unidades de dados de protocolo) que devem ser usados num caso de teste abstrato para aquele ATM.

Lower Tester (LT). Em todos os ATM's, o *lower tester* se comunica com o SUT através do fornecedor de serviço básico apropriado. O meio físico é considerado ser o fornecedor de serviço abaixo da camada física.

A especificação geral de ATM's refere-se a uma IUT na qual a camada mais superior é chamada de N_t (para *"top"*) e a camada inferior é chamada de N_b (para *"bottom"*). A mesma notação é usada para se referir as camadas dentro de um SUT e dentro do *lower tester*. O SUT pode implementar protocolos em camadas inferiores a N_b, porém estas camadas não são do

[6]Um SUT (System Under Test) é definido como um sistema composto de camadas de protocolo sob teste.

interesse da descrição do método de teste. Apesar disso, o SUT deve incluir a camada física. Para todos os métodos de teste, os ATS's especificam eventos de teste no ponto de controle e observação do *lower tester* em termos de $(N_b\text{-}1)$-ASP's e/ou N_b por meio de (N_t)-PDU's.

Upper Tester (UT). A principal diferença entre os ATM's está na natureza do *upper tester* e sua coordenação com o *lower tester*. Em alguns métodos de teste um segundo ponto de controle e observação é empregado para o *upper tester*. Nestes métodos a definição de eventos de teste no ponto de controle e observação do *upper tester* deve ser especificada de acordo com a definição de serviço OSI apropriada e com a especificação do protocolo OSI. A atividade no ponto de controle e observação do *upper tester* não exige que o SUT ou a IUT suportem parâmetros de primitivas de serviços abstratos, unidades de dados de protocolo ou capacidades que não são parte de um padrão internacional OSI ou uma recomendação do ITU. Se o ponto de controle e observação está numa interface *"humanamente"* acessível a interface do usuário do SUT deve servir como ponto de controle e observação.

Test Coordination Procedures (TCP). Para uma execução efetiva e realizável de testes de conformidade, algum conjunto de regras é exigido para a coordenação do processo de teste entre o *lower tester* e o *upper tester*. O objetivo geral dessas regras é habilitar o *lower tester* a controlar a operação do *upper tester*, de uma maneira necessária para rodar o pacote de teste selecionado para a IUT.

Estas regras levam ao desenvolvimento de procedimentos de coordenação de teste (TCP) para realizar a sincronização entre o *lower tester* e o *upper tester* e o gerenciamento de informações trocadas durante o processo de teste. Os detalhes dessa sincronização e como os efeitos exigidos são realizados estão intimamente relacionados com as características do SUT tão bem quanto com os métodos de teste.

2.5.1 - Metodologia de Teste para uma Única Camada

A ISO define métodos de teste abstratos para especificações de pacotes de teste, baseados na disponibilidade das ASP's nos pontos de controle e observação. Para teste de uma única camada, cada método é definido a seguir.

2.5.1.1 - O Método Local (L)

Este método é importante como uma base para um grande corpo da terminologia e conceitos, que são subsequentemente aplicados a outros modelos. A hipótese básica do método

local é que existem interfaces expostas abaixo e acima da IUT, estas interfaces servem como pontos de controle e observação, ou seja, pontos nos quais um sistema de testes pode controlar as entradas da IUT e observar as suas saídas.

Usando a convenção ISO, a camada sob teste é chamada de camada N e a próxima camada inferior de camada N-1. Definições de serviços de protocolo definem as primitivas de serviços abstratos (ASP's) trocadas nas interfaces do topo de uma entidade de protocolo. As ASP's incluem um conjunto lógico de eventos de teste (com parâmetros) que podem ser controlados e observados nas interfaces de uma IUT.

O conjunto de eventos de testes trocados na interface do topo são denotados como (N_t)-ASP's, esta interface deve ser uma interface de *hardware* padronizada que pode ser usada para propósitos de teste; os pacotes de teste não colocam qualquer exigência, adicional àquelas na especificação da interface de *hardware* padronizada, na realização da interface do SUT. A especificação de uma interface de *hardware* superior da IUT deve definir o mapeamento entre as ASP's e/ou PDU's relevantes e sua realização na interface. O mesmo é verdade para a interface inferior, e os eventos de testes são chamados (N_b-1)-ASP's. Cada protocolo define um conjunto de mensagens a serem trocadas com uma entidade par, elas são chamadas de PDU's (Protocol Data Unit). (N)-PDU's são trocadas como dados nos parâmetros de alguns das (N_b-1)-ASP's; outras (N_b-1)-ASP's carregam informações de controle para usar os serviços da camada N-1. O método de teste inclui dois elementos logicamente distintos que são chamados de *upper tester* e *lower tester* por causa de suas relações com as interfaces de uma IUT.

O *upper tester* é assumido para gerar e receber um conjunto de eventos de testes complementares ao conjunto de (N_t)-ASP's gerados e recebidos pela IUT na sua interface do topo, o *upper tester* está localizado dentro do sistema de teste. Hipóteses similares são feitas para a interface inferior da IUT e para o *lower tester*.

As regras do *upper* e *lower tester* são estimular a IUT através da troca de eventos de teste nas interfaces inferiores e do topo da IUT.

O *lower tester* está abaixo da IUT, funciona como uma entidade par do protocolo, trocando (N)-PDU's com a IUT através da sua interface inferior. Portanto, o *lower tester* pode emular o comportamento normal de uma entidade de protocolo da camada N durante teste, ou ele pode injetar erros para serem reconhecidos.

No método local, os procedimentos de coordenação de teste são expressos abstratamente pelas ASP's usadas para especificar as ações do *upper* e *lower testers*. As exigências para os procedimentos de coordenação de teste devem ser especificadas no ATS, mas são realizadas localmente dentro do sistema de teste.

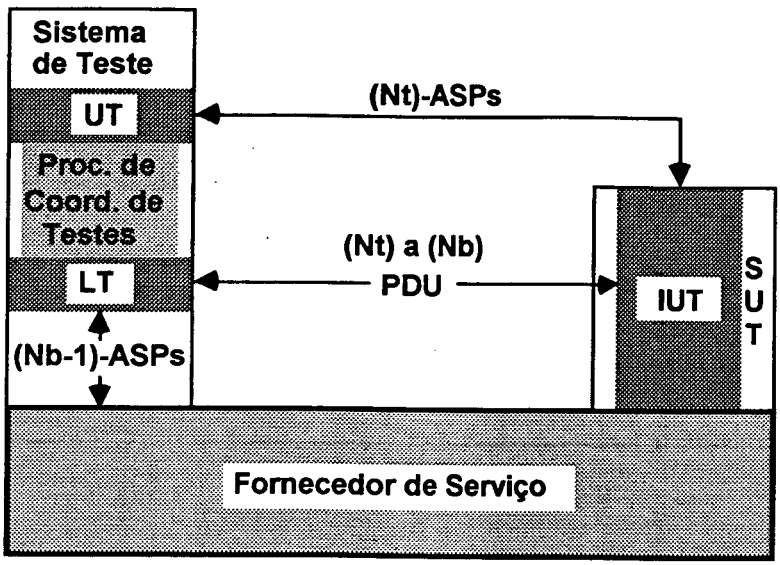

Figura 2.2. - O Método Local

Em resumo, o método de teste local assume uma armadura de teste em torno da IUT e expõe interfaces abaixo e acima da IUT, isto sendo ilustrado na figura 2.2.

O método local, embora simples, nem sempre é possível de ser aplicado (de fato, na maioria dos casos). Isto ocorre pois, quando um cliente chega em um laboratório de teste para testar uma implementação de protocolo, o acesso direto a interface inferior de uma IUT provavelmente não esta disponível.

2.5.1.2 - O Método Distribuído (D)

O método distribuído é um dos três modelos que não fazem hipóteses a respeito da existência de um ponto de controle e observação na interface inferior de uma IUT.

O *lower tester* e a IUT residem em dois sistemas diferentes. O *lower tester* e a IUT são conectados através de um serviço básico OSI, o qual oferece um serviço N-1 usando protocolos da camada inferior e o meio físico conectando os sistemas. Apesar do seu nome o *lower tester* é uma entidade par da IUT. O *upper tester* faz parte do modelo e na interface exposta no topo é assumido um ponto de controle e observação.

As principais conseqüências devido à mudança destas hipóteses são :

- casos abstratos de teste escritos para o método local não são aplicáveis, eles devem ser reescritos para refletir as primitivas de serviço abstratos disponíveis na interface inferior do *lower tester*;

- o *lower tester* e a IUT são separados fisicamente com a implicação de que eles observam o mesmo evento de teste em tempos diferentes;

- a perda de dados, entrega fora de seqüência e corrupção de dados são possíveis particularmente nas camadas inferiores, devido à qualidade do serviço oferecido nessas camadas;

- a sincronização e o controle são mais difíceis, porque elementos do sistema de teste estão distribuídos em dois sistemas.

Neste método os eventos de teste no ponto de controle e observação do *lower tester* são especificados apenas em termos de $(N_b\text{-}1)$-ASP's e/ou (N_b) por meio de (N_t)-PDU's e os eventos de teste no ponto de controle e observação do *upper tester* são especificados em termos de (N_t)-ASP's.

Neste método de teste o *upper tester* está localizado dentro do SUT e o limite do serviço superior da IUT deve ser tanto uma interface de usuário ou uma interface de linguagem de programação padronizada que pode ser usada para propósitos de teste. O pacote de teste não deve colocar qualquer restrição na realização da interface no SUT, adicional àquelas na especificação da interface da linguagem de programação padronizada, se aplicável.

Deve existir um mapeamento entre ASP's relevantes e sua realização na interface superior da IUT.

As exigências para os procedimentos de coordenação de teste devem ser especificadas nos ATS's. Se a interface superior da IUT é uma interface de usuário, então o operador do sistema que está sendo testado realiza as exigências dos procedimentos de coordenação de teste. Se a interface superior é uma linguagem de programação padronizada, então o *upper tester* é realizado em *software* e o *upper* e *lower testers*, juntos, realizam as exigências dos procedimentos de coordenação de teste.

Os ATS's para o método de teste distribuído não devem especificar uma interface para o *upper tester*. Para evitar a colocação de exigências no projeto interno dos sistemas que estão sendo testados, os ATS's não exigem que uma interface de linguagem de programação seja padronizada para o único propósito de teste.

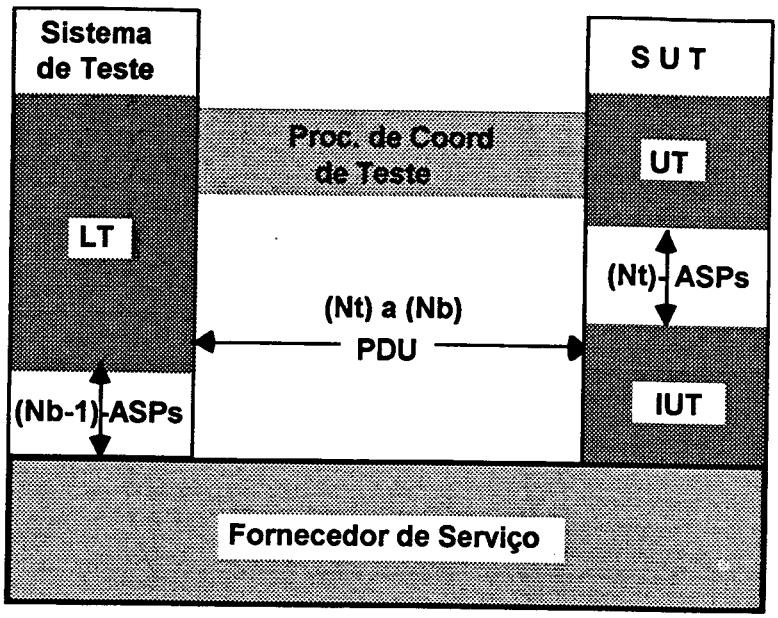

Figura 2.3. - O Método Distribuído

A sincronização e o controle (procedimentos de coordenação dos testes) podem ser especificados pelas primitivas de serviço abstratas trocadas nos pontos de controle e observação (ou possivelmente por um protocolo gerenciador de testes, que não é padronizado). O método distribuído confia no protocolo que está sendo testado para providenciar uma sincronização suficiente para concluir os propósitos de teste.

Em resumo, o método distribuído é logicamente equivalente ao método local, com o *lower tester* e a IUT interconectados por um serviço de comunicação, isto sendo ilustrado na figura 2.3.

Contudo, a estrutura do método local implicitamente fornece a capacidade de sincronizar e controlar o *upper* e o *lower tester*, porque eles são elementos da mesma armadura de teste. Então o método distribuído não é funcionalmente equivalente ao método local.

2.5.1.3 - O Método Coordenado (C)

Neste método os eventos de teste no ponto de controle e observação do *lower tester* são especificados apenas em termos de $(N_b\text{-}1)$-ASP's e/ou (N_b) por meio de (N_t)-PDU's somadas com as TM-PDU's. As (N_t)-ASP's não são usadas na especificação dos ATS's.

Os dois aspectos que distinguem o método coordenado do método distribuído são:

- nenhuma hipótese é feita a respeito da existência de um limite de serviço superior na IUT;

- um protocolo gerenciador de teste (TMP[7]) padronizado e uma unidade de dados do protocolo gerenciador de teste (TM-PDU) são usados para automatizar o gerenciamento dos testes e os procedimentos de coordenação. Muitas vezes é assumido que o *lower tester* é o mestre e que o *upper tester* é o escravo, para minimizar o esforço na realização do *upper tester*. O *upper tester* está localizado dentro do sistema que está sendo testado.

As exigências para os procedimentos de coordenação de teste devem ser especificadas no ATS por meio de um protocolo gerenciador do teste padronizado, que é referenciado através do ATS. O *upper tester* deve implementar o protocolo gerenciador do teste e realizar os efeitos apropriados na IUT.

Casos de teste devem ser somados com o ATS para o propósito de testar se o *upper tester* está conforme com as exigências da especificação do protocolo gerenciador do teste, tais casos de teste não contribuem com a avaliação de conformidade da IUT.

Um protocolo gerenciador do teste padronizado é aplicável a um ATS particular, padronizado para o método de teste coordenado e pode não ser aplicável a outros ATS's para o mesmo método de teste.

O protocolo gerenciador do teste deve ser implementado dentro do SUT, diretamente acima do limite de serviço abstrato no topo da IUT. Não deve ser exigido da IUT interpretar as TM-PDU's, ela deve apenas passá-las para e a partir do *upper tester*. O protocolo gerenciador do teste é definido apenas para testar um protocolo particular e deste modo não necessita ser independente do protocolo básico.

Os vereditos dos casos de teste não devem ser baseados na habilidade da sistema que está sendo testado em exibir qualquer primitiva de serviço abstrato ou parâmetros de uma primitiva de serviço abstrato no limite de serviço superior da IUT, desde que isso iria contradizer a definição do método de teste coordenado : o limite do serviço superior da IUT não é um ponto de controle e observação neste método de teste. Contudo, é recomendado que o protocolo gerenciador do teste seja definido separadamente a partir do ATS para facilitar a tarefa de implementação do *upper tester*.

Este é o método mais sofisticado, sendo ilustrado na figura 2.4, ele permite um alto grau de coordenação das informações observadas e coletadas em ambos *upper* e *lower tester*. O

---

[7]TMP (Test Management Protocol)

upper e o *lower tester* podem ser sincronizados, informações recebidas pelo *upper tester* podem ser mandadas de volta ao *lower tester* para verificação, seleções de casos de testes por um operador podem ser conduzidas do *lower tester* para o *upper tester* e ligações lógicas podem ser implementadas para a seleção de testes, se um teste anterior falhar.

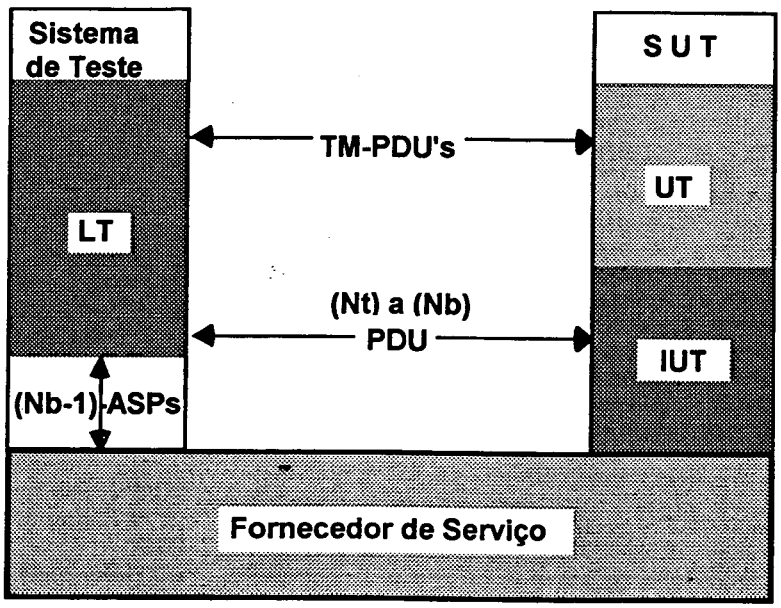

***Figura 2.4.** - O Método Coordenado*

As comunicações entre o *upper tester* e o *lower tester* podem ser *in-band* (as TM-PDU's são carregadas como dados através do protocolo que está sendo testado) ou *out-of-band* (usando um protocolo da camada inferior, o qual é assumido ter bastante confiabilidade para carregar as TM-PDU's).

2.5.1.4 - O Método Remoto (R)

Este é o último método definido pela ISO para testes de uma única camada, sendo ilustrado na figura 2.5. Este método serve para o caso onde não é possível observar e controlar o limite do serviço superior da IUT. As características significantes são que nenhuma interface é assumida no topo da IUT e nenhum procedimento de coordenação de testes é assumido. O método confia unicamente no protocolo que está sendo testado para a sincronização do *lower tester* e da IUT. O método assume que o estado de uma IUT é conhecido através das ações especificadas para o *lower tester*, incluindo o conhecimento das N-PDU's recebidas e transmitidas.

Neste método os eventos de teste no ponto de controle e observação do *lower tester* são especificados apenas em termos de (N_b-1)-ASP's, e/ou (N_b) por meio das (N_t)-PDU's. Os (N_t)-ASP's não são usados na especificação do ATS.

Algumas exigências para os procedimentos de coordenação de teste podem ser sugeridas ou expressadas informalmente no ATS, mas nenhuma hipótese deve ser feita a respeito da sua viabilidade e realização.

Abstratamente o sistema que está sendo testado necessita realizar algumas funções do *upper tester* para obter todos aqueles efeitos dos procedimentos de coordenação de teste e todo o controle e/ou observação da IUT que estão sugeridos ou expressados informalmente no ATS para um dado protocolo.

O *lower tester* deve tentar realizar os procedimentos de coordenação de teste sugeridos ou expressados informalmente de acordo com as informações relevantes contidas no PIXIT.

Para vencer a falta de especificação do comportamento acima da IUT, o comportamento exigido do sistema que está sendo testado deve ser especificado em termos das (N_b-1)-ASP's ou (N_b) por meio de (N_t)-PDU's que necessitam ser observadas pelo *lower tester*. Esta forma de especificação implícita deve ser tomada para fazer tudo aquilo que é necessário dentro do sistema que está sendo testado para provocar o comportamento exigido.

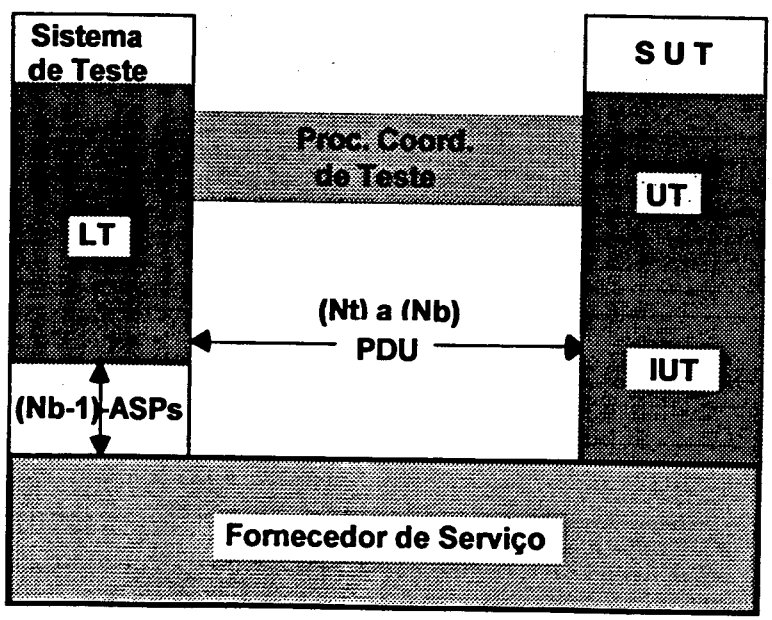

Figura 2.5. - O Método Remoto

Até mesmo com essa especificação implícita do controle da IUT, é possível neste método especificar o controle, mas não a observação acima da IUT. Esta é a maior diferença entre este método e os outros métodos de teste.

Os resultados devem ser formulados baseados sobre estímulos proporcionados pelo *lower tester* e as respostas da IUT quando observadas pelo mesmo.

Estes métodos de teste estão descritos em **[Bosik 91]**, **[Linn 90]**, **[Vuong 93]** e **[ISO9646-1]**.

2.5.2 - Metodologia de Testes Multi-camadas e Embutidos

2.5.2.1 - Os Métodos Embutidos

Os testes embutidos focalizam uma camada de protocolo de cada vez dentro de uma IUT que possui embutidas uma ou mais camadas dispostas numa pilha de protocolos.

Usualmente, o procedimento de teste é *bottom-up* dentro da pilha, as camadas superiores são testadas incrementalmente quando a conformidade das camadas inferiores é estabelecida.

Conceitualmente, os testes embutidos para duas camadas definem um conjunto de (N+1)-PDU's e então envolvem este conjunto de dados em (N)-PDU's apropriadas. A conseqüência direta destas hipóteses é que cada caso de teste deve especificar todas as ações das duas camadas (ou no mínimo um subconjunto funcional da camada N+1) para realizar o propósito de teste. Um caso deve ter um propósito de teste específico e deve refletir comportamentos dinâmicos possíveis para as duas camadas de protocolo da IUT.

A vantagem de um método embutido é que ele não assume uma interface em cada camada da IUT. Dada a existência dos métodos de uma única camada e os métodos embutidos, a ISO pode produzir pacotes de teste para qualquer combinação possível dos métodos local, distribuído, coordenado e remoto e grupos de protocolos.

2.5.2.2 - Os Métodos Multi-camadas

A OSI define os testes multi-camadas como *"Testar o comportamento de uma IUT multi-camadas como um todo, preferivelmente do que testar camada por camada"* **[Linn 90]**. De

qualquer modo, nenhuma informação adicional é dada neste tópico e atualmente o teste multi-camadas é assunto de pesquisa.

2.5.2.3 - O Método Ferry

Neste método o *upper tester* é recolocado como uma entidade, chamada *ferry control protocol*, o qual age como um *loop-back* lógico entre o *upper* e o *lower tester*. Os procedimentos de coordenação de testes, o *upper* e o *lower tester* podem residir no mesmo espaço de teste. O *ferry control protocol* direciona os (N)-ASP's entre o limite do serviço superior da IUT e o *upper tester* o qual está no final remoto da IUT.

O método ferry é uma versão distribuída do método local, onde a sincronização entre o *upper* e o *lower tester* é eliminada (eles estão no mesmo lugar). Contudo, ele requer um limite de serviço superior exposto e serviços *out-of-band* para o protocolo ferry.

Este método está descrito em **[Rayner 87]** e **[Linn 90]**.

2.6 - A Notação TTCN

Como os casos de teste devem ser a base para os testes de conformidade ISO/OSI, está claro que uma notação ou linguagem é exigida para especificar comportamentos de um sistema de teste e da entidade de protocolo a ser testada, esta linguagem deve conter aspectos suficientes para descrever testes para os protocolos OSI. Com estes propósitos em mente a ISO tem definido uma notação que é chamada TTCN (*Tree and Tabular Combined Notation*) **[ISO9646-3]**.

TTCN facilita a organização de elementos de teste dentro de pacotes de teste, isto é, uma coleção hieráquica de casos de teste. Um caso de teste, escrito em TTCN, é constituído de uma seqüência de eventos atômicos, os passos de teste, com isso nota-se que as expressões de comportamento são determinísticas, isto é, a ordem textual dos passos de teste num caso de teste é usada para resolver não-determinismos.

Uma especificação em TTCN consiste de uma *overview*, declarações, restrições e do comportamento dinâmico. A *overview*, escrita em inglês, descreve o escopo do pacote de teste. A declaração declara parâmetros do pacote de teste, pontos de controle e observação, unidades de dados de protocolo, primitivas de serviço abstrato e temporizadores. As restrições especificam valores para cada campo das unidades de dados de protocolo e primitivas de serviço abstrato.

As suposições básicas implícitas em TTCN são que uma entidade de protocolo numa IUT pode ser direcionada dentro de um estado assumido por uma seqüência de entradas e saídas (passos de teste). Uma vez atingido aquele estado, outra seqüência de passos de teste julga o comportamento de um aspecto particular da IUT e um veredito de pass/fail é formado. Finalmente a IUT é direcionada de volta para um estado inicial conhecido por ações subsequentes especificadas no caso de teste.

2.7 - Conclusões

Este capítulo abrange uma visão geral sobre os testes de protocolos de comunicação, principalmente no que diz respeito aos testes de conformidade.

Procurou-se detalhar todo o processo de testes definido pela ISO, apresentando-se a estrutura dos pacotes de teste de uma maneira que a compreensão deste processo seja facilitada.

Pode-se notar que as metodologias de testes são dependentes de diversos aspectos contidos nas implementações, tais como, disponibilidade de pontos de controle e observação, número de camadas da implementação do protocolo etc.

A estrutura e metodologia de testes de conformidade OSI levará a uma maneira para a produção de pacotes de testes padrão para protocolos OSI e para a harmonização das atividades de teste realizadas por diferentes laboratórios de teste, fornecedores e usuários.

CAPÍTULO 3

ESPECIFICAÇÕES FORMAIS X TESTES DE CONFORMIDADE

3.1 - Introdução

O processo de teste de um protocolo de comunicação é uma etapa importante e por isso não pode ser evitada. Esta etapa consiste em verificar se a implementação de um protocolo está conforme com a sua especificação. Isto é um problema difícil de ser solucionado, desde que o teste exaustivo das implementações de protocolos complexos (por exemplo, protocolos OSI) é teoricamente e praticamente improvável. Todavia a meta dos testes de conformidade é determinar tantos erros quantos possíveis numa implementação usando o mínimo número de casos de teste. Contudo, muitos protocolos de comunicação têm sido projetados e implementados sem levar em consideração os testes de conformidade, isto faz com que o esforço e o custo para se testar uma implementação sejam bastante grandes **[Beizer 90]**, então o conceito de testabilidade deve ser introduzido.

Testabilidade significa que um componente de *software* possui algumas propriedades, características ou aspectos que irão facilitar o processo de teste, tornando-se com isso uma propriedade que deve ser introduzida no projeto para as pessoas que irão testar o produto (um elemento de *hardware* ou *software*) e não, em geral, para os usuários finais. (dependendo da aplicação e/ou exigências, algumas facilidades de teste podem/devem ser deixadas disponíveis para o usuário final) **[Vuong 93]**. O projeto para testabilidade é o processo de aplicação de técnicas e métodos durante a fase de projeto para reduzir o custo e o esforço da fase de testes de uma implementação. A facilidade com que o processo de teste pode ser realizado depende de muitos fatores, por exemplo, ela depende das metas do teste ou da estratégia do teste (comprimento das seqüências de teste, tempo e cobertura dos testes), de como a IUT é vista : *black-box*, *gray-box* ou *white-box*, e o domínio ou aplicação do problema.

Os protocolos de comunicação têm duas características importantes que afetam os testes e a testabilidade, o não-determinismo e a coordenação. É importante notar que o não-determinismo e a coordenação não são exigências que devem ser introduzidas na especificação. Elas são simplesmente características da aplicação que está sendo manuseada.

O projeto de protocolos de comunicação deve ser realizado de uma forma a reduzir o custo e o esforço no teste da implementação. Para realizar esta meta é necessário se identificar o que constitue o projeto da especificação de um protocolo de comunicação e como ele pode ser representado, e, por outro lado, as metas e restrições de teste que serão impostas sobre a implementação.

Este capítulo mostrará na seção seguinte os principais problemas relacionados à testabilidade dos protocolos de comunicação, o não-determinismo e a coordenação e suas implicações no teste. Na seção 3.3 serão comentadas as técnicas de descrição formal, na seção 3.4 será dada uma breve descrição do uso dessas técnicas na implementação e, por fim, na seção 3.5 encontram-se as conclusões específicas deste capítulo.

3.2 - Problemas Associados à Testabilidade

3.2.1 - Não-determinismo

Existem três tipos de não-determinismo que podem ser identificados nos protocolos de comunicação envolvidos em sistemas distribuídos :

- não-determinismo devido a concorrência;
- não-determinismo devido a não-observabilidade;
- não-determinismo introduzido na especificação.

3.2.1.1 - Não-determinismo devido a concorrência

Um sistema distribuído pode ser visto como um conjunto de processos cooperantes, comunicando-se entre si através da passagem de mensagens. Em geral, não é possível determinar precisamente (isto é, deterministicamente) a ordem total dos eventos no sistema devido a falta de *clock's* globais sincronizados fisicamente. É apenas possível realizar alguma ordem parcial, e para alguns eventos não existe uma maneira de dizer qual deles ocorrerá primeiro. Este tipo de não-determinismo é inerente e inevitável em sistemas distribuídos.

3.2.1.2 - Não-determinismo devido a não-observabilidade

Se um módulo possui uma interface que não está diretamente acessível ou está embutida em outro módulo, seu comportamento, que será observado através de outros módulos, será descrito de maneira não-determinística. Até mesmo se a interface estiver diretamente acessível nem sempre pode ser possível observar o que acontece dentro do módulo, o que irá dar razão ao não-determinismo devido a não-observabilidade.

Este tipo de não-determinismo também é muito difícil de ser evitado, dado que é comum esconder detalhes de implementações atrás de alguns padrões de interfaces.

3.2.1.3 - Não-determinismo introduzido na especificação

O não-determinismo em protocolos de comunicação significa que quando um processo está num estado específico e recebe uma entrada particular, ele pode responder em duas ou mais maneiras diferentes, porém válidas. Não existe possibilidade de se prever qual resposta será oferecida (ao nível da especificação).

É claro que o não-determinismo pode ser evitado ao nível de especificação. Contudo, existem no mínimo três razões para se introduzir o não-determinismo na especificação : *concisão*, *desempenho* e *flexibilidade*. Atualmente o *desempenho* e a *flexibilidade* estão intimamente relacionados, porque, apenas será possível aumentar o desempenho se (de algum modo) o protocolo possuir algum grau de *flexibilidade*. Os aspectos de *desempenho* dependem da *flexibilidade*, porém, a *flexibilidade* pode facilitar outros aspectos (além do *desempenho*) tais como o custo operacional. A *flexibilidade* e o *desempenho* são ambos dependentes do ambiente no qual o protocolo é executado. O ambiente pode ser entendido como a rede (e seus parâmetros tais como velocidade e realização), *software* e *hardware*, aplicações etc.

Outra questão de *desempenho* está relacionada ao tempo. Não é desejável que decisões relacionadas ao tempo sejam tomadas ao nível da especificação. Por exemplo, o valor do *timeout* para retransmitir um pacote ainda não confirmado depende, entre outros fatores, do atraso de transporte que pode variar em diferentes ambientes.

A *flexibilidade* também é uma questão bastante importante, visto que o ambiente no qual o protocolo irá ser executado não é conhecido no momento do projeto. De fato, o mesmo protocolo pode ser implementado em muitos ambientes completamente diferentes. Então, algumas decisões não podem e não devem ser tomadas na especificação.

Com relação à *concisão*, quando se está especificando um protocolo, sempre existe mais de um evento válido que pode ocorrer num dado momento (estado). Para cada um desses eventos é sempre comum ter mais de uma ação válida que pode ser executada. Se o não-determinismo é introduzido para modelar os comportamentos válidos diferentes, não se torna necessário designar um estado diferente para cada par entrada/saída (isto é, evento/ação). Isto reduz o número de estados na especificação e facilita a compreensão do protocolo.

3.2.1.4 - Implicações no teste

Como foi visto, o comportamento não-determinístico significa que duas execuções do mesmo sistema podem produzir diferentes, mas contudo, válidas seqüências de eventos. Isto pode ser comparado com o teste de programas seqüenciais.

Em geral, para testar um programa seqüencial um conjunto de entradas de casos de testes é especificado e aplicado a uma implementação, e os resultados de teste são comparados com o comportamento esperado. Se um erro é detectado o programa é executado de novo (uma execução determinística) com a mesma entrada de teste para coletar alguma informação de depuração que irá ajudar a corrigir o programa. Uma vez que o erro é identificado, o processo de teste é repetido para verificar se o erro foi corrigido e se nenhum novo erro foi introduzido. Em softwares de comunicação (ou *softwares* concorrentes em geral), é muito mais difícil usar uma seqüência particular de eventos que irão levar a um erro, desde que existe uma quantidade enorme de tais seqüências. Também é mais difícil localizar a origem de um erro apenas analisando os resultados, por causa do não-determinismo. Sempre quando um erro é localizado e corrigido, é mais difícil executar o sistema de novo usando exatamente os mesmos passos que levam ao erro, isto é, a mesma seqüência de eventos. Além disso, também é muito difícil garantir que nenhum novo erro foi introduzido.

Outro problema é a geração dos casos de teste a partir de especificações contendo não-determinismo. Apenas recentemente trabalhos têm sido desenvolvidos na geração de seqüências de testes para protocolos modelados como FSM's não-determinísticas **[Arakawa 91]**, **[Fujiwara 91]**.

3.2.1.5 - Melhoras na Testabilidade

Na prática parece que as pessoas implementam protocolos de maneira determinística, uma vez que eles são mais fáceis de testar (porém o problema de geração de casos de teste ainda existe). Se existirem múltiplas escolhas, o implementador usualmente pega

uma de acordo com algum critério (por exemplo o conhecimento do ambiente no qual o protocolo irá rodar). Dar um objetivo particular ao teste poderia ser interessante para desenvolver métodos úteis na demonstração de que uma escolha particular eventualmente levará a este objetivo.

Quando existem múltiplas alternativas para o mesmo evento de entrada, o projetista deve ter certeza de que as diferentes escolhas são compatíveis entre si. Isto garante que diferentes implementações serão capazes de se comunicar. Novamente, seria desejável conceber mecanismos para mostrar se (ou não) diferentes escolhas são incompatíveis e como elas afetam o processo de teste.

Finalmente, seria também bastante útil para o projetista do protocolo ter alguma linha de orientação, como por exemplo *onde* e *porque* o não-determinismo deveria ser introduzido na especificação para diferentes tipos de protocolos.

3.2.2 - Coordenação

A coordenação é realizada por duas partes diferentes : *comunicação* e *sincronização*. A *comunicação* define as maneiras como os processos podem trocar informações. A *sincronização* define os mecanismos usados para ordenar os passos de computação entre processos no tempo.

3.2.2.1 - Comunicação

As seguintes três formas (ou paradigmas) de comunicação interprocessos podem ser usadas para definir primitivas de comunicação : troca de mensagens, chamada remota de procedimentos (RPC[8]) e transações. Estes mecanismos são sempre discutidos em livros que tratam de sistemas operacionais distribuídos e sistemas de gerenciamento de banco de dados **[Goscinski 91]**.

O mecanismo de comunicação mais usado nos protocolos é a comunicação através da troca de mensagens. Na troca de mensagens é assumido que as mensagens são enviadas e recebidas invocando as primitivas de comunicação do tipo SEND e RECEIVE, respectivamente. A

---

[8]Remote Procedure Call

0.224.908-8 Biblioteca Universitária UFSC

execução de uma primitiva RECEIVE depende da disponibilidade de uma mensagem. Se existir mais de uma mensagem disponível, é necessário determinar qual delas irá ser recebida. Isto pode ser feito tanto definindo algum critério ou recebendo uma delas randomicamente.

Uma primitiva SEND não-bloqueante, que permite ao emissor continuar a sua execução uma vez que essa primitiva foi enviada, é chamada comunicação *assíncrona*, enquanto que uma primitiva SEND bloqueante é chamada comunicação *síncrona*.

3.2.2.2 - Sincronização

Supondo que os protocolos de comunicação comunicam-se entre si apenas através de troca de mensagens e a sua execução é concorrente está clara a necessidade de um controle da execução de cada um desses protocolos em relação ao outro devido a sua própria aplicação (problemas de cooperação), ou para restringir o acesso a recursos comuns (problema de conflito). Na questão da cooperação, a evolução ou progresso de um processo depende somente do progresso de outro, enquanto que no problema do conflito um processo pode progredir mesmo que o outro não progrida. O conjunto de regras e mecanismos que definem e implementam tal controle é chamado *sincronização*. A necessidade para este controle é devido a ambos cooperação entre processos e competição por recursos compartilhados (sempre solucionados pelo sistema operacional).

Os problemas de sincronização aparecem porque os processos cooperantes não têm a mesma visão do seu estado global, desde que os processos são autônomos e as mensagens podem ter atrasos arbitrários. A análise de alcançabilidade, **[West 78]**, **[Zafiropulo 80]**, é sempre usada para detectar tais problemas. O problema com esta técnica é a *"explosão de estados"* (rápido crescimento do número de estados globais). Muitas técnicas têm sido propostas para reduzir o problema da explosão de estados, porém isto continua sendo o maior inconveniente da análise de alcançabilidade. Quando um problema de sincronização é encontrado, o protocolo deve ser reprojetado. Neste caso, o novo protocolo pode se tornar mais complicado e de difícil entendimento, ou pior ainda, novos problemas podem ser introduzidos. Isto leva a um ciclo de análise e modificação contínuo até que o protocolo esteja livre de erros.

Outro problema que pode afetar a sincronização é a perda da coordenação que pode ser causada por (entre outros fatores) uma inicialização inconsistente, erros de transmissão, falha e reconhecimento de processos e colisão de memória. Existe uma perda de coordenação quando os estados locais de cada processo (que podem ser corretos individualmente) formam um estado global inconsistente.

3.2.2.3 - Implicações no Teste

Os problemas de sincronização devido a coordenação são muito difíceis de serem identificados durante o processo de teste, uma vez que eles ocorrem durante situações particulares, tais como a combinação de eventos que levam a *deadlock* e perda de coordenação.

3.2.2.4 - Melhoras na Testabilidade

Um método alternativo para a análise de alcançabilidade é o projeto de protocolo que corrige a perda de coordenação. Esta técnica é chamada de *"auto-sincronização"*. Alguns pesquisadores propõem técnicas para construir protocolos que são *"auto-sincronizáveis"* baseados em CFSMs (*Communicating Finite State Machines*) **[Miller 87]**, **[Lin 89]**. Outros pesquisadores propõem técnicas usando FSM's (*Finite State Machines*) para construir protocolos *"auto-estabilizantes"*, isto é, protocolos que iniciam a partir de qualquer estado instável e convergem para um estado estável após a execução de um número finito de transições de estado, sem intervenção exterior **[Gouda 91]**.

Os algoritmos *"auto-estabilizáveis"* são algoritmos de sincronização, mas a *"auto-estabilização"* é uma propriedade do algoritmo. Isto torna a *"auto-estabilização"* um paradigma para projetar algoritmos de sincronização. Por exemplo, a transmissão assíncrona utilizando *start* e *stop bits* é *"auto-estabilizante"* independente de como o receptor inicia a recepção. Se o receptor inicia a recepção de uma maneira incorreta existirá um erro de *framing*. Eventualmente o receptor irá identificar um ou mais *stop bits* e irá se *re-sincronizar* (assumindo que o *hardware* está trabalhando corretamente).

3.3 - Técnicas de Descrição Formal

As FDT's[9] são essenciais em qualquer fase do processo de engenharia de protocolos. Em geral, as FDT's proporcionam uma base para :

1. o desenvolvimento de especificações não-ambíguas, claras e concisas;
2. a verificação de especificações;

---

[9]Formal Description Techniques

3. a análise funcional das especificações;

4. o desenvolvimento de implementações a partir de especificações;

5. a determinação de que uma implementação está conforme a sua especificação (isto é, teste de conformidade).

Para realizar os passos **1, 4 e 5**, uma FDT deve satisfazer as duas exigências básicas de poder de expressão e nível de abstração. O poder de expressão consiste na capacidade de uma FDT permitir aos seus usuários facilmente compor, examinar, entender, modificar e extender uma especificação. Isto inclui as exigências para concisão e facilidades de estruturação para especificações, por exemplo, composição de processos, abstração , instanciação e recursão. As exigências do nível de abstração dizem respeito a habilidade de uma FDT permitir a especificação dos conceitos da área de aplicação, isto é, os conceitos e construções da arquitetura OSI, no nível de abstração apropriado. Para realizar os passos **2-3**, uma FDT deve ter uma forte base matemática tornando-se fácil analisar e provar as propriedades semânticas e sintáticas desejáveis dos protocolos.

A ISO padronizou duas FDT's : ESTELLE **[IS9074]** e LOTOS **[IS8807]**, enquanto que o ITU padronizou uma FDT : SDL **[CCITT/Z-100]**.

3.4 - O Uso de Técnicas de Descrição Formal na Implementação

Algumas das questões que necessitam ser ponderadas quando se deseja obter uma implementação a partir de uma especificação formal são :

- o poder de expressão das linguagens de especificação e linguagens de implementação;
- a relação entre o estilo de especificação e implementação;
- as escolhas da implementação.

3.4.1 - O poder de expressão das Linguagens de Especificação e Linguagens de Implementação

Existem dois problemas que necessitam ser considerados quando se deseja mapear uma especificação formal dentro de um paradigma de implementação. Primeiro, dependendo do poder de expressão e do nível de abstração dos métodos formais, pode não ser possível implementar alguns aspectos do sistema. Segundo, dependendo da plataforma alvo e do

paradigma da implementação, pode ser difícil implementar ou representar certos aspectos da especificação, tais como, o não-determinismo, a comunicação, a sincronização, os temporizadores, etc.

O escopo da aplicação para todas as FDT's não deve ser restrito a fase de especificação de um protocolo de comunicação. A meta de todas as FDT's é suportar o ciclo total da engenharia de protocolos.

A aplicabilidade de cada FDT também irá depender da disponibilidade das ferramentas que irão ajudar os projetistas, entre outras funções, a introduzir testabilidade no projeto de uma maneira sistemática.

3.4.2 - A Relação entre o Estilo de Especificação e Implementação

A especificação deve ser escrita em um estilo que irá ser facilmente implementado e testado. As transformações de estilo devem ser definidas formalmente e ser automatizadas.

Idealmente, a implementação deve ser escrita de acordo com princípios definidos na engenharia de *software*, tais como, modularidade, separação de conceitos, antecipação de mudanças, generalidade etc [Ghezzi 91].

3.4.3 - As Escolhas da Implementação

Uma especificação formal não descreve qualquer detalhe da implementação. Por exemplo, no nível de especificação é possível ter uma situação onde dois ou mais eventos podem ocorrer, mas a semântica do método formal não diz qual evento irá ser tratado primeiro, pode ser o evento que ocorreu primeiro, ou um evento específico de acordo com algum critério, ou simplesmente um evento escolhido de uma maneira randômica. No contexto da metodologia e estrutura dos testes de conformidade OSI **[ISO9646-1]** esta informação é dada no PICS (*Protocol Implementation Conformance Statement*) e no PIXIT (*Protocol Implementation eXtra Information for Testing*), porém, as escolhas feitas ao nível da implementação devem ser "*visíveis*" durante o processo de teste para aumentar a testabilidade.

Também relacionado a este assunto está a questão dos limites físicos. Sempre, uma especificação formal não lida com os limites das variáveis, comprimentos das filas etc. Por exemplo, em ESTELLE o canal de comunicação é modelado como uma fila ilimitada. É claro que a nível de implementação os limites físicos devem ser impostos.

3.5 - Conclusões

Neste capítulo procurou-se mostrar a dificuldade de se testar uma implementação de um sistema derivada de uma especificação formal, nota-se que para facilitar o teste de um sistema de *software* ou *hardware*, uma exigência importante do projeto deveria ser a testabilidade.

O estudo de teste de um sistema distribuído em geral, e de protocolos de comunicação em particular, usando algum modelo é uma área relativamente nova de pesquisa. Muitos dos trabalhos que têm sido realizados em projeto para testabilidade no domínio de *software* **[Forghani 90]** estão relacionados ao projeto estrutural, todavia outros aspectos devem ser considerados.

Também é interessante notar que a nível de projeto é muito difícil separar as questões que afetam a verificação e o teste. Por exemplo, as escolhas não-determinísticas que podem ser introduzidas na especificação podem afetar a verificação (em termos do número de estados a serem alcançados) e o teste (em termos do número de comportamentos corretos que devem ser testados).

Idealmente, o processo de projetar protocolos de comunicação deveria compreender técnicas de síntese e de análise. O produto deste processo é o projeto de uma especificação que deve levar a uma implementação correta e facilmente testável.

No contexto de projeto para testabilidade a síntese denota o processo de construção de uma entidade usando primitivas ou elementos pré-definidos de acordo com algum algoritmo. A entidade final deve ter certas propriedades desejáveis para a construção. A análise denota o processo de composição de uma entidade em seus elementos e analisar estes elementos utilizando aum algoritmo. A meta deste processo é identificar algumas propriedades indesejáveis ou erros.

Os processos de síntese e análise devem ser baseados em métodos formais. Caso contrário, não seria possível definir precisamente ou provar *como* ou *se* as metas da síntese e análise podem ser realizadas aplicando certas regras e técnicas. Isto torna evidente, de novo, a importância da utilização de métodos formais no projeto de protocolos de comunicação.

CAPÍTULO 4

MÉTODOS DE GERAÇÃO DE TESTES A PARTIR DE ESPECIFICAÇÕES FORMAIS

4.1 - Introdução

O aumento da complexidade dos protocolos de comunicação, juntamente com o crescente número de equipamentos heterogêneos que devem ser interconectados entre si, têm destacado a necessidade para os testes de conformidade. Um ponto chave nesta questão é o projeto de pacotes de teste que são usados para realizar esta conformidade, desde que a eficiência de todo o processo de teste depende muito deste primeiro passo.

As técnicas de descrição formal têm desempenhado, nos últimos anos, um papel bastante importante no que diz respeito à concepção de protocolos de comunicação e mesmo de outras classes de aplicações distribuídas. Ferramentas de *software*, desenvolvidas em torno destas técnicas, permitem hoje automatizar as principais etapas de concepção de um protocolo ou de uma aplicação distribuída, então é explicável que a geração automática de pacotes de teste a partir de descrições formais de protocolos é atualmente uma área de pesquisa muito popular. Muitos métodos, baseados em vários modelos teóricos, têm sido propostos.

A geração (derivação) de seqüências de teste (isto é, seqüências de interações de entrada e possivelmente saídas esperadas) é uma tarefa de extrema importância para os testes de conformidade dos protocolos de comunicação.

A derivação de seqüências de teste para protocolos complexos é um processo bastante incômodo, considerando que todas as funcionalidades do protocolo devem ser checadas. Deste modo, fica evidente a necessidade de métodos automatizados para a derivação dessas seqüências.

A divisão deste capítulo traz, na seção seguinte, um resumo sobre os principais métodos baseados em máquinas de estados finitos onde são dadas as definições básicas necessárias para a aplicação destes métodos bem como o modelo de FSM utilizado. Na seção 4.3, são discorridos alguns métodos de geração de seqüências de teste baseados em técnicas de descrição formal, tais como ESTELLE e LOTOS, ainda nesta seção é dada uma breve descrição dessas técnicas. E, por fim, a seção 4.4 apresenta as conclusões específicas deste capítulo.

4.2 - Métodos Baseados em Máquinas de Estados Finitos

Uma técnica bastante utilizada para modelar um protocolo é um sistema de transição de estado. A especificação de um protocolo pode ser dividida em duas partes : a estrutura de dados e a estrutura de controle. Uma FSM[10] é um sistema de transição de estados simples que é tipicamente usada para modelar a estrutura de controle dos protocolos. Devido a estas razões, técnicas baseadas em FSM's têm sido largamente utilizadas na geração de seqüências de teste.

Segundo a literatura estes métodos podem ser divididos em quatro grupos : o método T (*Transition Tour*), o método D (*Distinguishing Sequences*), o método W (*Characterizing Sequences*) e o método UIO (*Unique Input/Output Sequences*) **[Bosik 91]**, **[Chow 78]**, **[Linn 90]**, **[Sabnani 88]**, **[Sarikaya 89]** e **[Sidhu 89]**.

4.2.1 - Definições Básicas

Antes de se descrever os métodos, algumas definições precisam ser apresentadas :

FSM determinística. Uma FSM é dita determinística se sua saída e o próximo estado estão em função do seu estado atual e da entrada que é aplicada. Para uma FSM determinística não podem existir dois ramos deixando um único vértice com a mesma operação de entrada, se isto acontecer, tais ramos, sem levar em consideração as suas operações de saída, implicarão que uma mesma operação de entrada pode causar duas transições de estados diferentes iniciando no mesmo estado, o que caracteriza um comportamento não determinístico.

FSM completamente especificada. Uma FSM é dita completamente especificada se os conjuntos de entradas permitidas para cada estado são equivalentes; qualquer entrada é permitida em qualquer estado e gera uma saída permitida, em outras palavras uma FSM é completamente especificada se a partir de cada estado ela possui uma transição para cada símbolo de entrada. Caso contrário a FSM é dita parcialmente especificada.

FSM fortemente conectada. Uma FSM é dita fortemente conectada se para cada par de estados (s_i e s_j) existe um caminho de transições indo de s_i para s_j.

---

[10]FSM (Finite State Machine)

Abaixo será apresentado o modelo de uma FSM utilizado na geração de seqüências de teste.

4.2.2 - A Especificação de Protocolos de Comunicação utilizando FSM's

Com estas definições a estrutura de controle de um protocolo é modelada como uma FSM determinística e é representada por um grafo G = (V,E), onde **V** é o conjunto finito de vértices (cada vértice corresponde a um estado da FSM) e **E** é o conjunto de ramos (cada ramo corresponde a uma possível transição de estados na FSM e possui um label **i/o** consistindo de uma operação de entrada **i**, **i** pertencente ao conjunto de operações de entrada **I** e é um evento que causa a transição de estado na FSM, e uma operação de saída **o**, **o** pertencente ao conjunto de operações de saídas O e podendo ser *null* ou invisível em alguns casos).

Para este modelo é assumido que a FSM possui um estado inicial (IS). A FSM pode alcançar o seu estado inicial a partir de qualquer estado por meio de uma entrada *reset* (*ri*/λ), durante esta transição a saída gerada é λ (representando a saída *null*), estas transições não são mostradas nesta figura, porém elas são necessárias para que a FSM seja fortemente conectada. No grafo G deve existir um caminho que alcançe cada um dos estados da FSM.

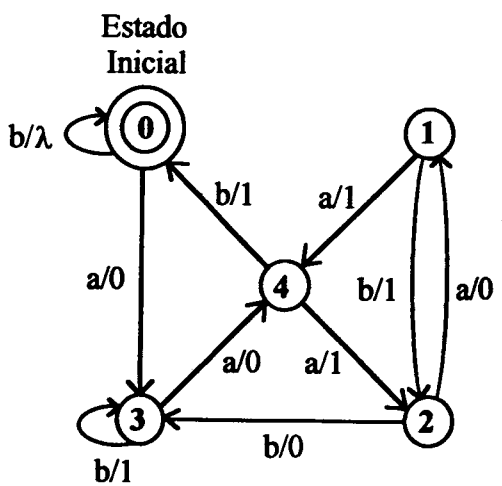

***Figura 4.1** - Exemplo de FSM*

No exemplo de FSM mostrada na figura acima a máquina possui duas entradas *a* e *b* e possui transições para ambas saídas de todos os estados exceto para o estado inicial 0. Esta FSM é feita completamente especificada pela adição do "*self loop*" b/λ para o estado inicial **0**.

Numa especificação de um protocolo, que é modelado como uma FSM determinística, testar uma extremidade (s_i; s_j; a_k / o_l) consiste de três passos :

1. colocar a implementação da FSM no estado s_i;

2. aplicar a entrada a_k e observar a saída o_l;

3. verificar se o novo estado da FSM é s_j.

Os passos acima para testar uma extremidade são de uma realização complicada, devido a limitações de *controlabilidade* e *observabilidade* da implementação da FSM.

A *controlabilidade* de uma implementação de protocolo é definida como a habilidade de levar a implementação para um estado desejável (através de um estímulo externo, tais como as entradas aplicadas a um estado) e a *observabilidade* é a habilidade de se observar o estado atual da implementação.

O passo 1 do teste, por causa da *controlabilidade* limitada, usualmente exige muitos passos (isto é, muitas operações de entrada/saída) para levar a implementação da FSM dentro do estado s_i. A menos que soluções eficientes sejam encontradas, limitações na *controlabilidade* podem facilmente resultar em seqüências de teste que são infinitamente longas.

As limitações na *observabilidade* previnem o testador de diretamente observar o estado da implementação do protocolo no passo 3 do teste, o qual é crítico para a capacidade do teste de detecção efetiva do erro.

Os quatro métodos, que serão apresentados a seguir, assumem que a estrutura de controle de um protocolo é modelada como uma FSM determinística, completamente especificada e que a implementação é fortemente conectada, esta última hipótese corresponde a dizer que a implementação está livre de *deadlocks* e *livelocks*.

Estes métodos serão apresentados abaixo.

4.2.3 - O Método T (Transition Tour)

Neste método, uma seqüência de teste é gerada baseada numa especificação de FSM, tal que, quando aplicada à implementação qualquer transição de estado definida pela especificação é testada. É óbvio que uma seqüência de teste gerada pelo método T apenas verifica a existência de transições e não testa os estados finais destas transições, porém, a grande vantagem deste método é a sua simplicidade quando comparado com os outros três métodos.

A principal desvantagem é que a questão da *observabilidade* não é tratada desde que os testes gerados por este método não executam o passo 3 do procedimento de teste.

Na tabela 4.1 é mostrada a seqüência de teste gerada pelo método T (junto com a seqüência de estados correspondente na linha posterior) para a FSM especificada na figura 4.1.

b	a	b	a	b	a	a	a	a	a	a	a	b	b
0	0	3	3	4	0	3	4	2	1	4	2	1	2

Tabela 4.1 - Seqüência de Teste para a FSM da figura 4.1

4.2.4 - O Método D (Distinguishing Sequences)

As seqüências geradas por este método (denominadas de seqüências DS) são definidas como um conjunto de entradas que gera um conjunto de saídas diferentes para cada estado da FSM. A seqüência de teste para a FSM a ser testada deve ser encontrada baseada na especificação. As principais desvantagens deste método são que em implementações típicas muito poucas FSM's possuem seqüências DS, e quando possuem essas seqüências são muito longas, tornando o uso deste método bastante limitado.

A menor seqüência DS para a FSM mostrada na figura 4.1 é *b,b*, visto que as saídas geradas por esta seqüência são diferentes para cada estado inicial da FSM. A tabela 4.2 mostra as saídas obtidas pela aplicação desta seqüência DS para cada estado da FSM.

Estado	Saída
0	λλ
1	10
2	01
3	11
4	1λ

Tabela 4.2 - Seqüência de Teste para a FSM da figura 4.1

4.2.5 - O Método W (Characterizing Sequences)

Para as FSM's que não possuem seqüências DS, o método de seqüências de caracterização define seqüências DS parciais, cada uma das quais distingue um estado específico da FSM de um subconjunto dos estados restantes, ao invés de distinguir este estado de todos os estados da FSM. O conjunto completo dessas seqüências de entrada para a FSM é chamado de conjunto característico *W* da FSM, este conjunto consiste de seqüências de entrada tal que os

últimos símbolos de saída observados a partir da aplicação dessa seqüência (em uma ordem fixada) são diferentes para cada estado da FSM. Este método é mais aplicável a implementações típicas do que o método D, visto que estas seqüências podem ser encontradas em todas as FSM's.

Para a FSM mostrada na figura 4.1 é fácil observar que $\{a, aa, b\}$ é o conjunto característico W. A tabela 4.3 mostra os últimos símbolos de saída para cada estado da FSM quando o conjunto de característico W é aplicado. Pela observação das seqüências de saída pode-se facilmente identificar cada estado da FSM.

Estado	saída (a)	saída (aa)	saída (b)
0	0	0	λ
1	1	1	1
2	0	1	0
3	0	1	1
4	1	0	1

Tabela 4.3 - Seqüência de Teste para a FSM da figura 4.1

4.2.6 - O Método UIO (Unique Input/Output Sequences)

Neste método uma seqüência de entrada é encontrada para cada estado s_i da FSM, denotada como UIO(s_i), tal que a saída gerada por UIO(s_i) é única para o estado s_i, ou seja este comportamento de i/o não é exibido por nenhum outro estado da FSM.

Estas seqüências UIO são usadas para gerar uma seqüência de teste que visita cada estado de uma transição. A seqüência UIO para um estado s_i pode apenas verificar que a FSM estava no estado s_i antes da seqüência UIO ser aplicada e não identifica o estado dessa FSM caso ela esteja em um outro estado que não seja s_i.

Formalmente uma seqüência UIO(s_i), é uma seqüência de entrada/saída especificada UIO(s_i) = $(i_1/o_1)(i_2/o_2)...(i_p/o_p)$ tal que não existe nenhum outro estado para o qual UIO(s_i) é uma seqüência entrada/saída especificada.

Este método focaliza o problema da *observabilidade* similarmente aos métodos das seqüências distintas e características. Contudo, ele tem maior preferência em relação aos outros métodos devido a duas razões :

- as seqüências UIO são tipicamente menores que tanto as seqüências distintas e características, desde que as seqüências de UIO são um subconjunto delas.

- quase todas as FSM's possuem seqüências de UIO, a menos que essa FSM tenha estados equivalentes.

A tabela abaixo mostra o conjunto de seqüências UIO para os estados da FSM especificada na figura 4.1.

Estado	UIO
0	b/λ
1	a/1 a/1
2	b/0
3	b/1 b/1
4	a/1 a/0

Tabela 4.4 - Seqüência de Teste para a FSM da figura 4.1

4.2.7 - O Método PW

O princípio deste método é a geração de dois conjuntos :

- o conjunto característico da FSM (conjunto *W*);
- o conjunto de seqüências ligadas com os caminhos parciais da árvore de teste (este conjunto é chamado de conjunto *P*).

Através da concatenação de *P* e *W*, todas as transições são disparadas. A FSM deve ser fortemente conectada e completamente especificada. Os caminhos resultantes são garantidos na detecção de qualquer mau comportamento da FSM.

4.2.8 - O Método PDS

Este método é executado para uma FSM tendo uma seqüência distinta (seqüência DS). As seqüências de teste são obtidas através da concatenação do conjunto P e das seqüências DS. Os caminhos resultantes são garantidos detectar todos os erros nas funções de saída e do próximo estado.

Uma descrição mais detalhada destes dois últimos métodos pode ser encontrada em **[Linn 90]** e **[Favreau 87]**.

4.3 - Métodos Baseados em Técnicas de Descrição Formal

O uso de técnicas de descrição formal para a especificação de protocolos de comunicação e mesmo outras classes de aplicações distribuídas tem suscitado a realização de trabalhos orientados à definição de métodos de geração de seqüências de teste a partir destas especificações.

Como foi mostrado no capítulo anterior as técnicas de descrição formal, tais como ESTELLE, LOTOS e SDL, oferecem um grande poder de expressão, porém se por um lado este poder de expressão permite ao programador um alto grau de abstração nos processos de síntese dos protocolos, aparece como um obstáculo nos processos de análise do mesmo.

Sendo assim, como será visto a seguir, a aplicação de métodos de geração de seqüência de testes impõe a realização de simplificações nos modelos adotados para estas técnicas, de modo que, sem perder totalmente o seu poder de expressão, as especificações sejam representadas numa forma mais adequada à aplicação de um algoritmo de geração de seqüências de teste.

4.3.1 - Geração de Seqüências de Teste a partir de ESTELLE

Antes de se começar a discutir os métodos será dada uma breve descrição da técnica ESTELLE.

ESTELLE é basicamente um modelo de EFSM[11] baseado num autômato estendido não-determinístico com a linguagem PASCAL e construções de estruturação adicionais **[ISO9074]**. ESTELLE modela um sistema especificado como uma estrutura hieráquica da instância do módulo do autômato que pode executar em paralelo, e pode se comunicar através de troca de mensagens e/ou compartilhando (de uma maneira restrita) algumas variáveis.

ESTELLE suporta o não-determinismo através de transições espôntaneas e permitindo a mais de uma transição, a partir de um estado principal, ter o seus predicados habilitados, a execução dessas transições é atômica. Num passo de execução uma escolha não-determinística é feita entre todas as atividades *"prontas para disparar"*.

---

[11] EFSM (Extended Finite State Machine)

Em ESTELLE a concorrência é implícita e pode ser *síncrona* ou *assíncrona* (usando a cláusula DELAY). O paralelismo assíncrono entre módulos é apenas possível entre subsistemas, enquanto o paralelismo síncrono é apenas possível dentro de subsistemas, isto é, uma instância de módulo não pode executar em paralelo com os seus descendentes. ESTELLE suporta apenas comunicação assíncrona entre módulos. Uma interação recebida por uma instância de módulo num dos pontos de interação é somada a uma fila FIFO ilimitada associada com aquele ponto de interação. A fila FIFO pode pertencer exclusivamente àquele ponto de interação (*individual queue*) ou pode se compartilhada com alguns outros pontos de interação de um módulo.

Para o processo de geração de seqüências de teste as construções relacionadas as transições são as mais importantes, estas construções são as cláusulas FROM, TO, WHEN, PROVIDED, BEGIN, OUTPUT.

As cláusulas FROM/TO definem o(s) estado(s) inicial/final da EFSM respectivamente. A chegada de uma interação de entrada é expressada usando a cláusula WHEN, transições sem esta cláusula são chamadas de transições espontâneas. As condições de disparo de uma transição são descritas na cláusula PROVIDED, que é uma expressão booleana em parâmetros de primitivas de interação e variáveis do módulo (chamadas de variáveis de contexto). Finalmente, a ação está contida em um bloco BEGIN que pode ter indicações para variáveis de contexto, chamadas para procedimentos internos, declarações CONDITIONAL PASCAL e procedimentos de saída com a declaração OUTPUT.

A linguagem ESTELLE, baseada numa EFSM, possui notações abstratas e concisas que a teoria atual para a geração de casos de teste automática não pode manusear. Então, transições definidas numa especificação ESTELLE serão convertidas em transições mais simples antes da geração dos casos de teste.

4.3.1.1 - Metodologia Baseada na Ferramenta de Geração de Teste CONTEST-ESTL

Esta metodologia toma uma especificação do protocolo de um único módulo normalizado em ESTELLE e deriva testes dos modelos dos grafos de fluxo de dados e controle **[Forghani 90]**. Especificações contendo múltiplos módulos são primeiro transformadas em especificações de um único módulo através da fusão de módulos.

Para derivar seqüências de teste os seguintes passos devem ser seguidos : primeiro uma referência formal da especificação do protocolo é escrita em ESTELLE. Esta especificação serve de entrada para a ferramenta de geração de testes chamada CONTEST-ESTL que realiza a

normalização e fusão dos módulos seguida pela análise do fluxo de dados e geração da seqüência de testes. As seqüências de teste resultantes são chamadas seqüências de testes não-parametrizadas dado que nenhum valor de parâmetro é determinado para entradas de teste.

A parametrização das seqüências de teste é difícil por causa das seguintes razões :

a. as cláusulas PROVIDED numa especificação ESTELLE podem ter qualquer complexidade, então podem existir várias maneiras de se satisfazer a cláusula dada.

b. numa dada seqüência de teste, um valor do parâmetro de entrada pode depender de valores determinados em transições anteriores.

As soluções para esta dificuldade são as seguintes :

- para o item a) deve-se transformar as cláusulas PROVIDED numa forma simplificada.
- para o item b) deve-se executar simbolicamente as transições numa seqüência de teste para determinar valores que serão levados de uma transição para outra.

As seqüências de teste (chamadas de *subtours*) obtidas da ferramenta CONTEST-ESTL são compostas de seqüências de transições no estado de maior ordem. Para uma dada implementação, durante a aplicação de tais seqüências, diferentes conjuntos de eventos podem acontecer devido ao não-determinismo. Por outro lado, casos de testes em TTCN permitem a especificação de comportamentos alternativos na forma de uma notação de árvore. Então as seqüências de teste lineares devem ser convertidas em árvores de teste considerando o comportamento não-determinístico das implementações. Tais casos de teste são chamados de *"must-test"*, desde que eles consideram todos os resultados possíveis de um dado teste.

A facilidade em ESTELLE na descrição de comportamentos não-determinísticos são as transições espontâneas. Estas transições jogam uma regra essencial na conversão de seqüências de testes para *"must-test"*.

O primeiro passo na obtenção de *"must-test"* parametrizados é a simplificação dos predicados de transições. Este passo é seguido da normalização e fusão do módulo feita pela ferramenta CONTEST-ESTL.

A normalização das transições é concluida movendo as condições nas declarações IF para cobrir um dos caminhos da cláusula PROVIDED. Cada transição corresponde a um caminho das declarações IF.

As transições normalizadas ainda não são adequadas para a geração automática de seqüências de teste. O problema é que o predicado da cláusula PROVIDED é complicado e existem muitas maneiras para satisfazê-lo. Durante os teste de conformidade todos os caminhos possíveis na satisfação desse predicado devem ser tentados.

Após a normalização, a cláusula PROVIDED é simplificada e é decomposta em muitos predicados atômicos simples, para qual existe uma única maneira de satisfação. O próximo passo no processo de simplificação é a criação de uma transição separada para cada expressão elementar.

Após o procedimento de simplificação, uma análise do fluxo de dados de parâmetros é feita, esta análise traça o fluxo de dados de parâmetros de primitivas de entrada (primitivas de serviço e/ou PDU's) para variáveis de contexto, e de variáveis de contexto para parâmetros de primitivas de saída sem considerar os predicados habilitados das transições normalizadas. O resultado é passado para uma ferramenta interativa chamada *dfgtool*, que mostra graficamente o fluxo de dados; esta ferramenta habilita o usuário para agrupar transições e identificar as funções do protocolo.

Usando outra ferramenta interativa chamada *testgen* o usuário pode obter seqüências de teste não-parametrizadas para cada função do protocolo. Neste ponto os cenários de teste são simplesmente uma seqüência de números de transições, as quais a IUT é esperada executar durante o procedimento de teste. Os cenários de teste devem ser convertidos em *must-test* e especificados como casos/passos de teste em TTCN. Isto é realizado em três passos : Primeiro, correspondendo a cada transição um passo de teste é criado. Segundo, transições espontâneas são combinadas com transições WHEN. Terceiro, estes passos de teste são unidos com a árvore principal na ordem especificada no cenário de teste dando o caso de teste. O passo final é a geração dos casos de teste.

Na geração dos casos de teste todas as declarações na ação são processadas e convertidas para possivelmente um ou mais eventos de recepção TTCN cada vez em níveis de identificação incrementados. Qualquer designação a variáveis de contexto são convertidas a declarações TTCN na ordem em que elas aparecem. Todavia, essas variáveis de contexto devem ser definidas como variáveis globais TTCN na parte de declaração.

Após os passos de teste serem criados, as transições espontâneas são tratadas. As transições espontâneas cujos estados iniciais são os mesmos que as transições em consideração e têm declarações de saída, se tornam alternativas para as transições para as quais a sub-árvore está sendo construída.

As declarações de saída são transladadas da mesma maneira, mas o nível de identificação dos eventos gerados é tal que esses eventos se tornam alternativas para o último evento gerado.

Em resumo a metodologia descrita consiste de primeiro transformar a especificação original para outra especificação ESTELLE mais simples. Após a simplificação as seqüências de teste são geradas de uma maneira semi-automática, finalmente estas seqüências são parametrizadas e convertidas automaticamente para uma notação TTCN. Isto é feito mapeando-se as diferentes construções especificadas em ESTELLE para as suas construções correspondentes em TTCN.

4.3.1.2 - Metodologia Baseada na Ferramenta LISE

Neste método **[Favreau 87]** um compilador é usado para determinar se os atributos estáticos da especificação do protocolo estão corretos e produz :

a. um código C para ser usado na implementação do protocolo.

b. uma descrição de máquina abstrata modelando o protocolo.

Uma máquina abstrata pode ser definida por :

- S : um conjunto finito de entradas;
- I : um alfabeto finito de entradas;
- O :um alfabeto finito de saídas;
- V : um conjunto finito de variáveis;
- A : um conjunto finito de ações em variáveis;
- F : um conjunto de funções booleanas nas variáveis (predicados).

Transição :

Com este formalismo uma transição é definida como $< q_0, i, f, q'', o, act >$, onde:

- q_0 é o estado atual;
- i é uma entrada de I

- *f* é uma função booleana de *F*;
- *q"* é o próximo estado;
- *o* é uma saída de O;
- *act* é uma ação de A.

Uma máquina abstrata (EFSM com predicados e variáveis) não tem a mesma estrutura de uma FSM, ela reconhece seqüências de entrada, troca estados baseado nas entradas e predicados, tem memória (variáveis) e executa ações, incluindo emissões de saídas durante transições entre estados.

A máquina abstrata é especificada num formalismo que pode ser diretamente usado por uma ferramenta de *software* específica. Esta ferramenta é chamada LISE e foi projetada na Universidade de *Bordeaux I*, França, e no A.D.I (*Agence de l'Informatique*, Paris, França). O programa LISE é composto de duas ferramentas, a ferramenta VADILOC e a ferramenta GAST.

A ferramenta VADILOC executa as tarefas de verificação de algumas propriedades locais do autômato, e a validação da comunicação entre duas entidades que usam o protocolo através de uma meio físico com capacidades finitas (canais limitados). As possibilidades da ferramenta VADILOC são as seguintes :

- coleção de transições da máquina abstrata;
- verificação de propriedades locais (conectividade, existência de caminhos entre dois estados etc);
- validação da comunicação entre dois autômatos através do método da perturbação baseado na análise da acessibilidade.

A ferramenta GAST é usada para gerar seqüências de teste. Os métodos do roteiro de transições, PDS e PW são implementados para gerar seqüências de teste de uma FSM. A geração de seqüências de teste de uma máquina abstrata é realizada através do método de desenvolvimento do contexto e um método PW adaptado para máquinas abstratas. A ferramenta GAST é usada juntamente com a ferramenta VADILOC.

4.3.1.3 - Metodologia Baseada no Modelo TOF (Test Oriented Form)

Uma maneira de se gerar casos de testes a partir de uma EFSM é simplificá-la numa FSM, ignorando todas as variáveis internas e parâmetros de entrada e ignorando todas as condições habilitadas (isto é, assumindo que elas sempre são satisfeitas) **[Chun 90]**.

Converter uma EFSM numa FSM pura equivalente é um caso extremo de normalização. Em vez disso, um estado de uma EFSM pode ser decomposto num número menor de estados com um certo alcance, no qual o predicado de uma transição pode ser satisfeito. Os estados decompostos, consistindo de um nome de estado e uma condição sobre uma variável domínio, são chamados de *minor states*. O procedimento de geração dos casos de teste trata os *minor states* como estados diferentes. As entradas decompostas, consistindo de um nome de interação da entrada e uma condição sobre um domínio do argumento de entrada, chamados eventos também são tratados como entradas diferentes se as condições de ambas entradas estão desarticuladas. TOF é uma intermediária entre uma FSM pura e ESTELLE generalizada. Este modelo assume que um estado de uma EFSM pode ser decomposto num pequeno número de estados com um certo alcançe no qual o predicado de uma transição pode ser satisfeito. Por exemplo, assumindo que existem *m* transições de saída de um estado *src* quando uma entrada *ip* é aplicada, estas transições são definidas como $t_i = (src, dst_i, ip, cnd_i, blk_i),\ 1 \le i \le m$ e $R_i \subseteq dom(v_1)$ x ... x $dom(v_n)$ ser o subdomínio das variáveis $v_1,...,v_n$ no qual cnd_i pode ser satisfeita. Então o estado *src* pode ser decomposto num conjunto de estados $< src, R_1 >$ ... $< src, R_k >$. Uma interação de entrada pode ser decomposta numa maneira similar com $< ip, P_1 >$...$< ip, P_k >$ onde $P_i \subseteq dom(a_1)$ x ... x $dom(a_k)$ é o subdomínio dos parâmetros de entrada $a_1,...,a_k$ no qual cnd_i pode ser satisfeita. Ambos R_i e P_i são condições necessárias para cnd_i, são representadas como funções $R_i = f(v_1,...,v_n)$ e $P_i = f(a_1,...,a_k)$ respectivamente.

Desta maneira, um estado pode ser decomposto em um número muito menor de estados, sendo que o procedimento de geração de seqüências de teste trata estes estados como estados diferentes.

Então uma EFSM é considerada estar em TOF se todas as suas transições satisfazem as seguintes definições :

Uma transição $t \in T = (minor, dst, event, cnd, Olist, Vlist)$ está em TOF se e somente se :

1. cada estado *minor* possui exatamente uma transição de saída e é definido como uma tuple <*src,R*> onde *src* é o nome do estado principal e **R** é uma condição sobre uma domínio variável;

2. o estado destino *dst* é o mesmo que o da especificação original;

3. a interação de entrada *event* é definida como uma tuple de <*ip*,*P*> onde ip é o nome da interação e *P* é uma condição sobre um domínio do argumento de entrada;

4. *cnd* é definida como uma condição sobre ambos um domínio do argumento de entrada e um domínio variável, as condições *minor*, *event* e *cnd* são expressadas na forma conjuntiva. A forma conjuntiva é uma lista de termos lógicos conectados pelo operador "Λ (*and*)". Cada termo lógico é uma constante booleana (*t* ou *f*), ou um par de termos aritiméticos conectados por operadores relacionais tais como =,≠, <,≤,>,≥;

5. *Olist* é uma lista de interações de saída;

6. *Vlist* é uma lista dos valores das variáveis internas em termos dos valores anteriores que são definidos sobre um domínio finito.

A conversão de qualquer aspecto ESTELLE no seu equivalente TOF é um problema complexo. Para simplificar este problema algumas restrições são feitas. É assumido que um protocolo é especificado em ESTELLE com uma única EFSM que é completamente especificada, isto é, não existem tipos indefinidos e primitivas de funções ou procedimentos. Cada procedimento e função é assumido ser completamente especificado sem chamadas recursivas. Num corpo de procedimento ou função, valores podem ser determinados apenas para variáveis locais ou parâmetros passados através de chamadas de procedimentos.

Também é assumido que o número de interações para todos os *loops* é conhecido estaticamente antes da execução. Desde que muitos protocolos têm blocos de transições relativamente simples, isso pode ser assumido sem perda de generalidade. Um projetista de teste pode modificar o bloco de transições decompondo-o em múltiplos blocos de transições sem declarações *loop* ou com declarações *loop* que tem interações contáveis estaticamente, se o protocolo tem *loops* não-determinísticos estaticamente no seu bloco de transições.

A EFSM é assumida ser fortemente conectada, isto é, qualquer estado pode ser eventualmente alcançado a partir do estado inicial.

Para obter-se um modelo TOF a partir de um conjunto limitado ESTELLE, deve-se normalizar o bloco de transições, o qual tem várias declarações que podem causar configurações entrada/saída diferentes. Para normalizar uma transição, o primeiro passo envolve a transformação das seguintes declarações num bloco de transições.

- declarações CONDITIONAL são eliminadas através da decomposição da transição em uma ou mais transições, cada uma delas representa um caminho de controle distinto da

declaração CONDITIONAL. A cláusula PROVIDED é modificada para refletir a condição do caminho de controle.

- declarações LOOP são eliminadas através da repetição do corpo da declaração LOOP para qualquer valor da variável indexada. É assumido neste ponto que o número de interações pode ser determinado estaticamente.

- declarações WITH são usadas para gravar variáveis *acess*. Cada declaração WITH pode ser eliminada anexando a estrutura WITH no início de cada arquivo *acess* dentro do escopo da declaração.

- declarações de chamadas de procedimento/função são trocadas pelo corpo do procedimento/função correspondente. Na troca de uma declaração chamada, cada parâmetro é simbolicamente trocado por um parâmetro atual.

- declarações ESTELLE tais como *initialize*, *attach*, *detach*, *connect*, *disconnect* etc, não são permitidas.

O próximo passo é transformar cada transição em TOF.

A cláusula PRIORITY em ESTELLE tem significado apenas quando múltiplas transições estão habilitadas. Se apenas uma transição está habilitada, ela será disparada sem levar em consideração a sua prioridade. Para propósitos de teste, a cláusula PRIORITY pode ser eliminada anexando uma condição adicional à condição habilitada, tal que apenas a transição a ser testada torna-se disparável e todas as outras de prioridade superior são desabilitadas.

Por causa da sua definição semântica, a cláusula ESTELLE DELAY(min,max) não pode ser usada para especificar (ou testar) restrições de tempo real, com isso, hipóteses razoáveis devem ser feitas para testar implementações reais.

Todavia, para propósitos de teste a cláusula delay pode ser tratada como um tipo especial de evento de entrada. Do ponto de vista de teste, se a transição dispara antes da unidade de tempo *min* ou se ela dispara após a unidade de tempo *max*, a implementação da IUT possui falhas.

Basicamente, a geração de testes de conformidade para um modelo de TOF é similar a geração de testes para um modelo de FSM pura. Ambos os modelos podem ser representados como grafos diretos onde estados são representados por nós e transições são representadas como pontes entre pares de estados. Da mesma maneira que o modelo de uma FSM, o modelo TOF também gera seqüências de testes compostas de três partes :

1. um caminho para colocar uma IUT no estado fonte especificado src_t da transição t;

2. uma transição t;

3. uma seqüência UIO para verificar o estado destino especificado dst_t da transição t.

Contudo, o modelo TOF difere de um modelo de FSM no sentido de que o modelo TOF é uma FSM estendida. Apesar disso, as extensões do modelo TOF são mais restritas do que uma EFSM geral, o modelo apenas tem fluxo de dados tão bem quanto fluxo de controle. Então, o mecanismo de travessia para o grafo representando um protocolo modelado em TOF deve ser modificado para manusear as extensões. Igualmente no mesmo estado e com a mesma interação de entrada, um conjunto de próximas transições possíveis pode variar dependendo do contexto corrente, tal como valores de variáveis internas ou parâmetros de entrada. Isto significa que deve-se manter uma informação sobre o contexto corrente e qualquer troca de contexto após cada transição ser executada.

Este método de geração dos casos de teste envolve problemas complexos tais como a execução simbólica e soluções de restrições. O problema de solução de restrições teoricamente não possui uma solução, contudo, ele pode ser reduzido para um problema que apresenta solução com o acréscimo de algumas restrições, isto só pode ser realizado por causa que muitos predicados numa especificação de um protocolo são simples o suficiente, tal que sem muita perda de generalidade pode-se restringir o escopo apenas para relações num domínio finito.

O modelo proposto gera casos de teste efetivos que cobrem o fluxo de dados e controle completamente por que ele decompõe uma transição com múltiplos caminhos de execução em muitas transições, cada uma delas é coberta por um caso de teste distinto.

Em resumo os métodos de geração de seqüências de teste baseados em ESTELLE se constituem dos três passos descritos abaixo :

1. aplicar um conjunto de transformações para a especificação original de modo a obter uma forma normalizada (simplificada);
2. converter a forma normalizada (simplificada) num modelo direcionado para a geração de seqüências de teste;
3. gerar seqüências de teste baseadas neste modelo.

Um outro exemplo de um modelo pode ser encontrado no método descrito em **[Ural 87]**, neste o modelo utilizado é o *NFS* (Normal Form Specification), neste método antes de se derivar seqüências de teste é feita uma análise do fluxo de dados da especificação.

A meta desta análise é detectar práticas de codificação questionáveis (ou anomalias no fluxo de dados) num dado código fonte. As anomalias no fluxo de dados mais comuns são : referenciar variáveis indefinidas, definir variáveis sem uso subsequente, etc. Além disso a DFA[12] traça o fluxo de dados dos parâmetros de primitivas de entrada (primitivas de serviços e/ou PDU's) para variáveis de contexto e de variáveis de contexto para parâmetros de primitivas de saída sem considerar os predicados habilitados das transições normalizadas.

Outros métodos de geração de seqüências de teste baseados em ESTELLE podem ser encontrados em **[Lee 92]** e **[Phalippou 90]**.

4.3.2 - Geração de Seqüências de Teste a partir de LOTOS

Antes de se começar a discutir os métodos será dada uma breve descrição da técnica LOTOS.

LOTOS tem uma definição formal completa baseada em dois paradigmas : um sistema de transição para expressar o comportamento dinâmico na parte do processo, e uma definição de tipo abstrato de dados baseada em ACT ONE na parte de dados **[ISO8807]**. LOTOS modela um sistema distribuído como um processo, que pode ser composto de muitos sub-processos. Um sub-processo é também um processo. Então, em LOTOS um sistema é expressado como uma hierarquia de processos. Ações internas a um processo não são observáveis. Um processo pode interagir com outros processos, os quais formam seu ambiente, através de ações observáveis.

Em LOTOS a concorrência é explícita. LOTOS suporta apenas comunicações síncronas entre processos. Toda comunicação (interação) com o ambiente é feita através de pontos de interação denominados *gates*. Uma interação pode ocorrer quando dois ou mais processos estão prontos para executar a mesma ação observável. Uma ação observável consiste de um oferecimento/aceitação de zero ou mais valores num gate.Uma interação pode envolver troca de dados, e é um evento instantâneo, isto é, comunicação síncrona. O não-determinismo é um aspecto básico de especificações em LOTOS e pode ser explicitamente introduzido tanto por um operador *[ ]* ou através de um evento interno ***i***.

---

[12]DFA (Data Flow Analisys)

4.3.2.1 - Metodologia Baseada no Modelo CHART

A metodologia aqui apresentada é constituida de um algoritmo que converte um subconjunto de FULL-LOTOS em CHART, que é um tipo particular de um sistema de transições. Neste método o sistema CHART pode ser considerado como um passo intermediário para a derivação de testes a partir de especificações em LOTOS, este modelo está formalmente definido em **[Tripathy 91]**.

Como nos métodos baseados em ESTELLE, são aplicadas algumas transformações na especificação original, estas transformações têm como objetivo simplificar a especificação antes de convertê-la num modelo orientado para geração de seqüências de teste. As principais transformações aplicadas a especificação original para a obtenção do modelo CHART são comentadas abaixo :

- transformação de um operador seqüencial (>>) num operador paralelo semanticamente equivalente;
- transformação de um operador de sincronização total num operador paralelo generalizado, esta transformação é feita inserindo-se todos os gates sincronizados dentro do operador paralelo no qual os processos sincronizam (||[*gates sincronizados*]||).
- remoção dos eventos internos guardados.

Os testes gerados a partir do modelo CHART são não-lineares, devido ao não-determinismo natural deste modelo.

Para a geração dos casos de teste, uma avaliação dos predicados deve ser feita. Após esta avaliação os predicados em CHART são classificados em três classes:

Predicados determináveis : Nos protocolos de comunicação existem dois tipos de predicados determináveis : os predicados estáticos e os predicados dinâmicos.

Os predicados estáticos são aqueles que dizem respeito às variáveis cujos valores não mudam. Estes tipos de predicados são gerados em ***chart*** devido ao tipo da comparação dos valores das interações na solução da composição paralela. Os predicados dinâmicos são aqueles que dizem respeito às variáveis cujos valores são trocados por uma cláusula ACTION ou designação da regra seguindo um evento específico.

Predicados não-determinísticos : Estes predicados não podem ser completamente controlados nem pela IUT e nem pelo testador. Os predicados não-determinísticos são aqueles que dizem

respeito às variáveis cujos valores são gerados de uma maneira não-determinística. Isto acontece no caso da geração de valores, por exemplo numa construção generalizada. Na prática, esses valores podem ser proporcionados no PICS do fornecedor da implementação OSI.

Predicados dependentes : Predicados que dependem de valores dos parâmetros das primitivas de entrada pertencem a esta categoria. Estes predicados podem ser colocados verdadeiros através da escolha dos valores apropriados para os parâmetros dos eventos de teste.

4.3.2.2 - Metodologia baseada em LOTOS e TTCN

O modelo apresentado aqui mapeia especificações em LOTOS e TTCN para EFSM e ajusta todas as atividades do teste de conformidade. O comportamento de um protocolo e de uma entidade de teste é descrito como um conjunto de transições, cada uma acionada por uma entrada, uma saída ou um evento interno. Os dados são descritos em tipos abstratos de dados.

A ESFM CIM é uma sextupla : $M = <J, V, j_0, j_f, R, \varepsilon>$, onde J é o conjunto de estados, V é o conjunto de declarações de dados (variáveis), j_0 é o estado inicial, j_f é o conjunto de estados finais, R é o conjunto de transições e ε é o conjunto de designação de valores iniciais para as variáveis em V.

Converter uma especificação LOTOS em um pacote de testes TTCN envolve:

1. converter a parte do comportamento de um especificação LOTOS numa EFSM.
2. converter as declarações e restrições TTCN para tipos abstratos de dados.
3. converter o comportamento dinâmico TTCN para uma EFSM.

A porção de dados numa especificação LOTOS já é descrita em tipos abstrato de dados, com isso não necessita ser convertida.

A primeira etapa do processo é converter uma especificação em LOTOS para uma EFSM este passo consiste em :

- normalizar a especificação em LOTOS através da transformação de um operador de habilitação num operador paralelo semanticamente equivalente.
- converter a especificação normalizada para uma EFSM através da aplicação de uma regra de transformação para cada operador LOTOS. Conceitualmente, cada regra de transformação deriva todos os comportamentos seqüenciais possíveis de dois

comportamentos operantes referidos pelo operador. As regras de transformação são necessárias apenas para operadores que não foram eliminados durante a normalização. Existe um algoritmo de transformação implementado em *Prolog* **[Naik 92]**.

A notação TTCN usa um número pré-definido de tipos de dados, como *integer*, *boolean*, *bit string*, *hex string*, *octet string* e *character string*. Os casos de teste têm duas classes de tipos de dados estruturados, frequentemente usados : as primitivas de serviço abstrato e as PDU's. Em qualquer arquitetura de teste, o tester e a entidade de protocolo comunicam-se entre si através da troca de primitivas de serviço abstratos, algumas dessas primitivas podem conter PDU's. Em TTCN, as primitivas de serviço abstrato e as PDU's estão numa forma tabular. Para existir a capacidade de comparação entre valores de dados gerados por um caso de testes com os valores admitidos por uma especificação de protocolo, deve-se converter a descrição tabular de primitivas de serviço abstrato e PDU's dos casos de teste em TTCN em tipos abstratos de dados.

Para converter uma PDU num tipo abstrato de dados, deve-se definir um operador construção para construir a PDU, um operador para identificar o tipo da unidade da PDU construída, e um operador seleção para selecionar cada campo da PDU.

Pode-se derivar algoriticamente uma EFSM da parte dinâmica de um caso de teste em TTCN. A conversão consiste da expansão do comportamento default, derivando uma EFSM da árvore principal e de cada sub-árvore, e resolver a ligação das sub-árvores através da união das EFSM's correspondentes. Cada linha de eventos de um caso de teste é modelada como uma transição numa notação de uma EFSM.

Após a especificação em LOTOS ter sido convertida numa EFSM, pode-se derivar casos de teste da EFSM gerada. O primeiro passo é gerar algoriticamente estruturas de controle dos casos de teste, estas estruturas são chamadas estruturas dos casos de teste. O segundo passo é transformar manualmente as estruturas dos casos de teste em casos de teste TTCN completos. O número de estrutura dos casos de teste derivadas depende do número de transições e do número de estados da EFSM do protocolo.

Se um modelo de protocolo é determinístico, então cada caso de teste consiste de uma seqüência de eventos de teste derivados de alguma técnica de geração de testes. Se um modelo de protocolo é não-determinístico, não se pode representar um caso de teste com uma simples seqüência de eventos. Para corretamente julgar o comportamento de tais protocolos, um caso de teste deve atentar para o comportamento alternativo esperado.

Então, o algoritmo de geração de teste consiste dos seguintes passos. A entrada é a EFSM gerada da especificação LOTOS, e as saídas são as estruturas dos casos de teste na forma de EFSM.

1. na EFSM dada, gerar uma nova seqüência de transições começando e terminando no estado inicial, esta seqüência é chamada de caso de teste parcial.

2. na EFSM se existir uma transição *r* tendo um evento interno que é uma alternativa para a transição do caso parcial de teste, então deve-se atualizar o caso parcial de teste somando a seqüência de transições tal que a seqüência começe em *r* e termine em um estado pertencente ao caso parcial de teste. Repetir o passo 2 ou ir para o passo 3.

3. o caso de teste parcial é uma estrutura de casos de testes, se todos as estruturas foram geradas então pare. Caso contrário ir para o passo 1.

Para se derivar um significativo caso de teste de uma estrutura de caso de teste, deve-se modificar o comportamento da estrutura do caso de teste para encontrar um propósito de teste e somar dois outros comportamentos, O *Timeout* e o *Anormal*. Nos sistemas em tempo real, a noção de *timeout* é bem conhecida, mas não se pode derivar automaticamente informações do tempo de uma especificação de protocolo. Um caso de teste deve ter um evento *Anormal* como uma alternativa para qualquer evento de recepção pegar eventos inesperados de uma implementação incorreta.

Quando um caso de teste é transformado em TTCN, todas as transições de entrada e saída no caso de teste da EFSM são transformados em transições de entrada e saída em TTCN. Uma seqüência de eventos no caso de teste de uma EFSM é transformada em uma seqüência de eventos indentados, com cada evento indentado representando as progressões de tempo. Um conjunto de transições alternativas é representado como um conjunto de linhas de eventos alternativos. Os loops numa EFSM são propriamente representados por *loops goto* em TTCN. Um veredicto *Pass* é designado para o evento final no caminho pretendido, e um veredicto *inconclusivo* é designado para os outros caminhos derivados da EFSM do protocolo. Veredictos de falha são designados para os eventos *Anormais.*

Um outro método de geração de seqüências de teste baseado em ESTELLE está descrito em **[Pitt 90]**.

4.4 - Conclusões

Neste capítulo foram apresentados alguns métodos de geração de seqüências de testes a partir de especificações formais (FSM's e FDT's).

Nota-se que a questão de geração de seqüências de teste baseada no modelo de máquinas de estados finitos depende de algumas características dessas máquinas (FSM

completamente especificada, fortemente conectada etc) fazendo com que o projeto da especificação do protocolo obedeça estas a características.

Para a geração de seqüências de testes a partir de técnicas de descrição formais nota-se que todos os métodos apresentados, tanto para LOTOS quanto para ESTELLE, executam algum tipo de transformação na estrutura da especificação. Estas transformações podem ser normalizações (simplificações) e/ou eliminação de alguma cláusula, além de serem aplicadas algumas restrições à especificação. Um outro ponto comum é a definição de *modelos intermediários* a partir dos quais pode-se gerar seqüências de testes. Nota-se que estes modelos diminuem o poder de expressão das FDT's para com isso reduzir o problema do não-determinismo.

Foi notado que todos os métodos de geração de seqüências de testes estudados, tanto para geração a partir de FSM's quanto para geração a partir de FDT's (ESTELLE e LOTOS) têm como limitação principal o problema do não-determinismo, tanto que todos esses métodos de geração estão baseados em modelos o mais determinísticos possíveis.

CAPÍTULO 5

UMA METODOLOGIA PARA TESTE DE PROTOCOLOS ESPECIFICADOS EM ESTELLE E FSM

5.1 - Introdução

É comum encontrar, na literatura, resultados evocando métodos de geração de seqüências de teste a partir de especificações formais, sejam elas representadas utilizando máquinas de estado ou técnicas de descrição formal como LOTOS e ESTELLE.

No que diz respeito às técnicas de descrição formal, e como descrito no capítulo anterior, a totalidade dos métodos apresentados impõem a necessidade de simplificações preliminares nas especificações formais. Estas simplificações têm por objetivo eliminar parte do alto poder de expressão destas técnicas que, se por um lado, favorecem, em muito, a tarefa de especificação de sistemas complexos tais quais os protocolos de comunicação, introduzem dificuldades incontornáveis à aplicação de algoritmos de geração de seqüências de teste. Algumas destas questões foram discutidas no capítulo 3.

O método aqui apresentado, apesar de requerer, similarmente, simplificações na especificação original, propõe um escalonamento maior nestas simplificações, sendo que boa parte destas podem ser automatizadas, até que, pela aplicação de um método orientado às máquinas de estados finitos, é possível obter as seqüências de teste.

O interesse pela técnica ESTELLE é justificável não só pela disponibilidade de ferramentas que auxiliam a verificação e validação de protocolos de comunicação (ou de sistemas distribuídos de uma maneira geral) especificados utilizando esta técnica mas também devido aos trabalhos que já foram desenvolvidos, tanto a nível da UFSC, no LCMI (Laboratório de Controle e Microinformática) abordando esta técnica **[Manhas 94]**, **[Leite 92]**, **[Silveira 91]** e **[Willrich 91]**, como a nível internacional **[Courtiat 87a]**, **[Budkowski 87]** e **[Diaz 89]**.

As seções que seguem apresentarão as diferentes etapas deste método, partindo de uma especificação formal utilizando a técnica de descrição formal ESTELLE.

A divisão deste capítulo traz no item seguinte uma descrição da metodologia de geração de seqüências de teste proposta, desde a especificação do protocolo até a geração das

seqüências de teste através da ferramenta implementada. Na seção 5.3 é apresentada a ferramenta de geração de seqüências de teste implementada, na seção 5.4 são mostrados alguns exemplos de aplicação da metodologia proposta. Na seção 5.5 são discorridas as restrições que devem ser observadas para o uso desta ferramenta. Por fim, a seção 5.6 apresenta as conclusões relativas a este capítulo.

5.2 - Descrição da Metodologia

Intuitivamente, o método baseia-se na obtenção, a partir de uma especificação formal, de uma máquina de estados que represente o comportamento observável[13] do sistema a ser testado, no caso uma entidade de protocolos. Obtida a máquina de estados, o conjunto de seqüências de teste é obtido pela aplicação de um método de geração definido para esta categoria de modelos.

Sendo assim, podemos estruturar o método aqui proposto segundo as etapas abaixo:

1. transformação da especificação formal;
2. obtenção da máquina de estados representando o comportamento observável da entidade de protocolo;
3. transformação da máquina de estados;
4. geração da seqüência de testes.

As seções a seguir vão descrever, em detalhes, o princípio associado a cada uma destas etapas.

5.2.1 - Transformação da Especificação Formal

Nesta primeira etapa do método, a principal tarefa é a preparação da especificação formal para a etapa a seguir. Apesar de ser esta a única etapa não automatizada do processo de geração de testes, ela não impõe grandes dificuldades nem reduções significativas no poder de

[13]Por comportamento observável entende-se a máquina de estados composta por um conjunto de transições, cujas etiquetas representem eventos observáveis, ou seja, eventos ocorrendo nas interfaces externas do sistema sob observação. No caso de uma entidade de protocolos, estes eventos representam, normalmente, primitivas de serviços (e/ou, em alguns casos, unidades de dados de protocolos — PDUs) trocadas entre a entidade e as demais componentes do sistema.

expressão da técnica de descrição formal utilizada, o que vem a ser, sem dúvida, uma grande vantagem deste método.

A tarefa de preparação da especificação formal consiste em traduzir, manualmente, a especificação formal descrita em ESTELLE (versão ISO) para uma nova especificação descrita em ESTELLE*. ESTELLE* é uma versão estendida de ESTELLE caracterizada por um maior poder de expressão graças à introdução de um novo mecanismo de comunicação (o *rendez-vous*) e outras modificações na semântica original de ESTELLE. Para maiores detalhes sobre esta versão, recomendamos consultar **[Courtiat 87a]** e **[Courtiat 87b]**.

A justificativa para a transformação de uma especificação ESTELLE para outra descrita em ESTELLE* será apresentada na seção seguinte. Sendo assim, nos limitaremos a apresentar as principais modificações a serem feitas na especificação original para a obtenção de uma especificação em ESTELLE*.

Estas modificações apresentam-se em duas categorias: as modificações no mecanismo de comunicação de ESTELLE (filas FIFO) para o mecanismo introduzido na versão ESTELLE* (*rendez-vous*); as modificações estruturais na especificação ESTELLE, que serão caracterizadas pelas modificações nos atributos da especificação e pela introdução de "módulos de ambiente" para o módulo que descreve o comportamento da implementação a ser testada (a entidade de protocolo).

5.2.1.1 - Modificações no Mecanismo de Comunicação

As transições da especificação em ESTELLE que apresentem cláusulas de recepção (WHEN) na sua parte pré-condição deverão ter estas cláusulas substituídas por outras de sincronização em recepção como definidas em ESTELLE*; assim, para transições que apresentem a forma :

```
TRANS
   WHEN ip.interação
     begin
     ...
     end;
```

a modificação vai refletir-se na obtenção de uma transição na forma:

```
TRANS
   ip?interação
     begin
     ...
     end;
```

Para este tipo de transformação, as demais cláusulas (PROVIDED, por exemplo) são mantidas inalteradas, assim como as instruções contidas no bloco *begin ... end* da transição (com exceção da instrução OUTPUT de ESTELLE).

As transições contendo a instrução OUTPUT em seu bloco *begin ... end* deverão também ser modificadas, de modo que uma transição da forma :

```
TRANS
  ...
    begin
      ...
      ...
      OUTPUT ip.interação
    end;
```

ficará, após a transformação, na forma :

```
TRANS
  ip!interação
    begin
      ...
      ...
    end;
```

Com relação a estas transformações, devemos considerar dois casos típicos a serem resolvidos que podem aparecer frequentemente em especificações formais :

1. *transições incluindo o processamento de parâmetros a serem enviados num OUTPUT* : não são poucos os casos em que os parâmetros de uma interação a ser enviada são calculados na própria transição; neste caso, duas transições deverão ser geradas a partir desta transição: a primeira, sem cláusula de sincronização, contendo um bloco *begin ... end* caracterizado pelas instruções que permitem calcular os parâmetros da interação a ser enviada; a segunda, caracterizada unicamente por uma cláusula de sincronização em emissão, explicitando os parâmetros a serem enviados na comunicação; esta segunda transição pode conter, eventualmente, em seu bloco *begin ... end*, instruções que não estejam associadas à computação dos parâmetros da interação a ser enviada. Exemplo :

   transição original (ESTELLE)

```
TRANS
  FROM S1 TO S2
    begin
      p1 := funcao1(i1,i2);
      p2 := função2(i3,i4);
      OUTPUT ip.inter(p1,p2);
    end;
```

transições obtidas (ESTELLE*)

```
TRANS
   FROM S1 TO X
     begin
       p1 := funcao1(i1,i2);
       p2 := funcao2(i3,i4);
     end;

   FROM X TO S2
     ip!inter(p1,p2)
       begin
       end;
```

No exemplo, é possível notar a definição de um novo estado, adicional, X, que permite estabelecer a atomicidade da execução seqüencial das duas transições no contexto do módulo.

2. *transições contendo emissão e recepção simultânea* : no caso, muito comum, de uma transição apresentando simultaneamente cláusula de recepção e instrução de emissão (*output*), a transformação impõe, similarmente ao primeiro caso exposto, a explosão de uma transição em duas. Verificar o exemplo abaixo :

transição original (ESTELLE)

```
TRANS
   FROM S1 TO S2
     WHEN ipA.interacao1
     begin
       OUTPUT ipB.interacao2;
     end;
```

transições obtidas (ESTELLE*)

```
TRANS
   FROM S1 TO X
     ipA?interacao1
     begin
     end;

   FROM X TO S2
     ipB!interacao2
       begin
       end;
```

Pela mesma razão explicada anteriormente, notamos aí o surgimento de um estado adicional na especificação do módulo considerado. Aos estados surgidos para o propósito da especificação, vamos dar o nome de *estados artificiais*. Faremos referência a esta classe de estados, na descrição de uma das etapas posteriores.

5.2.1.2 - Modificações Estruturais da Especificação

A primeira das modificações a serem introduzidas na especificação consiste na transformação dos atributos ESTELLE ("*systemactivity*", "*systemprocess*", "*activity*" e "*process*") utilizados para definir o modo de paralelismo a ser imposto à implementação. Independente do atributo considerado na especificação, sua configuração final deverá ser caracterizada pela associação do atributo "*systemactivity*" à especificação e "*activity*" aos demais módulos que a compõem. Esta, apesar de ser uma restrição imposta para o uso da técnica ESTELLE*, permite cobrir, sob o ponto de vista do paralelismo, uma grande parcela dos possíveis comportamentos do sistema especificado.

A outra modificação importante é a introdução (ou a substituição, se for o caso) dos "*módulos de ambiente*" que servirão de interface externa ao(s) módulo(s) que descrevem o comportamento da entidade de protocolos, cujas implementações serão objeto de teste de conformidade. A definição destes módulos é necessária, ao contrário de outras técnicas, uma vez que uma especificação ESTELLE deve ser escrita na forma de um "*mundo fechado*", desprovido portanto de interfaces com o "*exterior*".

Por outro lado, do ponto de vista do comportamento, estes módulos de ambiente, apresentam-se relativamente simples, dado que eles servem de acessório para as comunicações entre o módulo que representa a entidade de protocolo sob teste e o ambiente com o qual a implementação vai interagir.

A figura 5.1 sugere a forma mais comum da arquitetura de uma especificação ESTELLE*, construída para a geração das seqüências de teste. Pode-se notar na figura, a presença do módulo que representa a entidade de protocolo, assim como os módulos de ambiente introduzidos na especificação, representando o usuário do serviço provido pela entidade e o serviço de comunicação utilizado por esta.

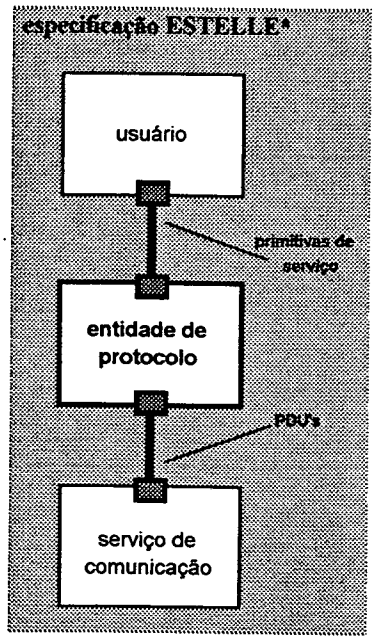

***Figura 5.1** - Forma geral de uma especificação ESTELLE* para a geração de seqüências de teste*

Os corpos representando o comportamento destes módulos de ambiente vão, na maior parte dos casos, ser caracterizados pela existência de um único estado global, a partir do qual (e para o qual) serão disparadas todas as transições. Estas, por sua vez, serão caracterizadas, na sua totalidade, por cláusulas de sincronização (*rendez-vous*) em emissão ou em recepção, sendo que as interações vão representar as mensagens trocadas entre a entidade de protocolos e o seu ambiente: as primitivas de serviço e as unidades de dados de protocolo (PDUs). As transições devem, então, representar todas as possíveis interações de comunicação entre a entidade de protocolo e o seu ambiente, ou seja, todas as trocas de unidades de dados de protocolos e ou primitivas de serviço que irão ocorrer durante o funcionamento do protocolo.

A parte ação das transições (bloco *begin ... end*) poderá ser totalmente vazia ou caracterizada por funções ou procedimentos de cálculo dos parâmetros das interações.

Com relação à declaração da máquina de estados propriamente dita, dado que esta é caracterizada pela existência de um estado único, não existe necessidade da utilização da declaração STATE, que permite definir os estados globais de cada corpo de módulo. A limitação no que diz respeito às possíveis interações podendo ocorrer entre o módulo que representa a entidade de protocolo e os demais módulos (ordenação de primitivas de serviço e PDUs) será controlada unicamente pela máquina (pelas máquinas) de estado representando a entidade (uma vez que esta é caracterizada pela existência de estados globais e outras componentes - variáveis, cláusulas PROVIDED, etc... - que permitem adicionar restrições de comportamento a esta máquina) e pelo uso do mecanismo de comunicação por *rendez-vous*.

A ilustração do uso de ESTELLE* para a especificação de uma entidade de protocolos será vista mais adiante, neste capítulo.

5.2.2 - Obtenção da Máquina de Estados representando o comportamento observável da Entidade de Protocolos

A máquina de estados que representa o comportamento observável da entidade de protocolos vai ser o objeto de aplicação da técnica de geração de seqüências de teste, técnica esta orientada ao modelo máquinas de estado finito.

A obtenção desta máquina de estados, portanto, requer o processamento da especificação formal descrita em ESTELLE de modo a obter todas as possibilidades de execução do módulo (ou dos módulos) representando a entidade de protocolos, na forma de um conjunto de estados e de transições etiquetadas pelas possíveis interações na forma de primitivas de serviço e de PDUs, interações estas que comporão as seqüências de teste que serão geradas posteriormente.

Por outro lado, a máquina de estados gerada deverá "*esconder*" a totalidade do comportamento interno da especificação da entidade de protocolos, traduzida em ESTELLE por transições expontâneas nos módulos (transições sem cláusulas ou instruções de comunicação) ou transições associadas a comunicações internas (entre os módulos compondo a especificação da entidade de protocolo).

Para isto, vamos utilizar as facilidades de verificação existentes no ambiente ESTIM (*ESTELLE SimulaTor based on an Interpretative Machine*) desenvolvido no LAAS-CNRS (França), **[Saqui 89]**, e que permite obter o grafo de alcançabilidade de uma especificação de protocolo descrito em ESTELLE*. Esta é a razão pela qual a transformação de uma especificação formal em ESTELLE para ESTELLE* é uma etapa necessária dentro da metodologia proposta.

O grafo construído pela ferramenta de verificação ESTIM representa todas as possibilidades de execução da especificação formal submetida, sendo que este vai apresentar normalmente duas classes de transições representando o comportamento :

- transições etiquetadas por interações externas, que são aquelas associadas à troca de interações entre o módulo (ou módulos) representando a parte da especificação que está sob análise;

- transições etiquetadas por *null*, que representam aquelas transições relacionadas ao comportamento "interno" do módulo sob análise.

Para a obtenção da máquina de estados representando o comportamento observável da entidade de protocolo a ser testada, é necessário ainda, eliminar a segunda classe de transições descrita acima. Esta eliminação é realizada de forma computacional, pela aplicação de relações de equivalências entre autômatos, sendo que, para isto, adota-se a equivalência de rastro. Esta equivalência permite eliminar, do grafo gerado por ESTIM, todas as transições internas, gerando um autômato (dito autômato quociente) caracterizado unicamente pelas transições etiquetadas pelas interações externas.

A automatização do método de redução do grafo por equivalência de rastro está presente num módulo da ferramenta PIPN (*Prolog Interpreted Petri Nets*), também desenvolvida no LAAS-CNRS, **[Lloret 90]**, com o qual o ambiente ESTIM é interfaceado.

Ainda neste capítulo, será mostrado, através de um exemplo, como estas ferramentas são utilizadas neste contexto.

5.2.3 - Transformação da Máquina de Estados

Esta etapa é constituída de duas partes :

1. Transformação do arquivo gerado pelo ESTIM numa FSM contendo todos os estados artificiais;

2. Transformação para a forma de entrada da ferramenta implementada.

5.2.3.1 - Transformação do arquivo gerado pelo ESTIM numa FSM contendo todos os estados artificiais

Nesta etapa o arquivo de saída do grafo de acessibilidade gerado pelo ESTIM com a extensão *.proj.lab.l.auto* (encontrado no anexo III) é transformado para a forma de uma FSM intermediária, que é guardada num arquivo com a extensão *.tim*. Esta transformação consiste em se ler o arquivo gerado pelo ESTIM, onde as seguintes informações devem ser retiradas : o estado inicial da transição (ou ramo), a interação de entrada desse estado, a interação de saída desse estado e o estado final da transição. Quando um estado não possui a interação de entrada ou a interação de saída, então ela deve ser preenchida com a interação *null*, representando uma entrada ou saída nula.

Um exemplo do arquivo gerado pelo ESTIM e do arquivo gerado após esta primeira transformação pode ser encontrada no anexo III.

5.2.3.2 - Transformação para a forma de entrada da ferramenta implementada

Nesta etapa a FSM intermediária gerada pela transformação do arquivo gerado pelo ESTIM sofre uma nova transformação. Esta transformação tem como objetivo eliminar os estados artificiais introduzidos durante a etapa de especificação em ESTELLE*.

A eliminação dos estados artificiais é realizada através da fusão de transições ($a f b$, sendo a e b duas transições), da seguinte maneira :

1. é verificado se o estado de chegada da transição a é igual ao estado de saída de b, com esta regra tem-se que o estado de saída de ($a f b$) é igual ao estado de saída da transição a e o estado de chegada de ($a f b$) é igual ao estado de chegada de b;

2. a apresenta um *label não nulo/nulo* e b apresenta *label nulo/não nulo*, com esta regra tem-se que o label de ($a f b$) é igual ao label de a unido com o label de b;

3. b é a única transição de saída do seu estado de saída.

O resultado desta fusão é a FSM que representa o fluxo de controle do protocolo especificado, esta FSM é armazenada num arquivo com a extensão *.fsm* que servirá de entrada para a ferramenta de geração de seqüências de teste implementada.

Um exemplo deste arquivo pode ser encontrado na seção 5.4.

5.2.4 - Geração das Seqüências de Teste

O método de geração de seqüências de teste implementado detecta todas as faltas se o número de estados na implementação for igual ao da especificação. É assumido que a estrutura de controle do protocolo é modelada como uma FSM determinística, mínima e completamente especificada. A implementação é vista como uma caixa preta (*black box*) com uma porta de entrada e uma porta de saída, possuindo duas interfaces, uma interface com o usuário (ou com o protocolo da camada superior) e uma interface com o protocolo da camada inferior. O método utiliza a *"Completeness Assumption"* que diz o seguinte : a entidade de protocolo ignora uma entrada aplicada num estado para o qual a saída e o comportamento do próximo estado não estão especificados no *Core Behaviour* (comportamento desejável do sistema, este comportamento pode não cobrir todas as combinações possíveis de entrada/saída).

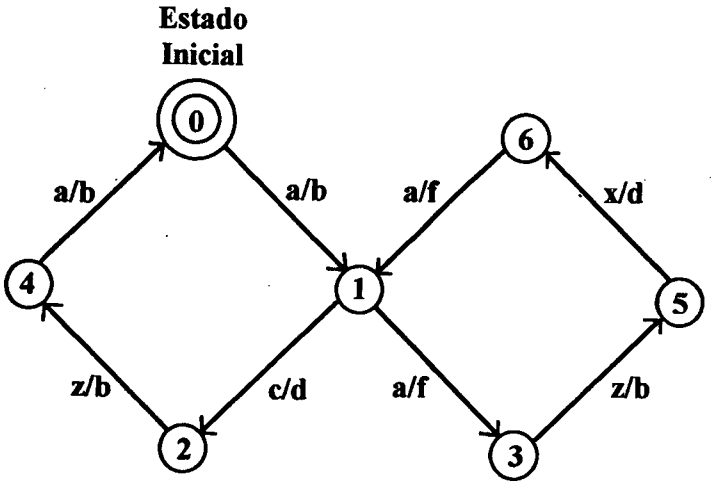

Figura 5.2 - Exemplo de FSM.

Por exemplo para a FSM mostrada na figura 5.2 o conjunto dos ramos desejáveis é : {(01), (12), (24), (40), (61), (56), (35), (13), (00), (10), (20), (30), (40), (50), (60)}.

O modelo de FSM utilizado é aquele apresentado no capítulo anterior, obedecendo todas as restrições impostas a este modelo.

Para cada ramo (transição de estado) $e \in E$ são definidas três funções : *Head*(e), *Tail*(e) e *Label*(e), onde *Head*(e) denota o estado inicial para *e*, *Tail*(e) denota o estado final de *e*, e *Label*(e) denota o label de *e*.

Neste método uma seqüência é definida como uma série de pares entrada/saída. Em termos formais uma seqüência S é dada como : $S=(i_1/o_1)(i_2/o_2)...(i_k/o_k)$ onde $i_p \in$ I e $o_p \in$ O com p sendo um inteiro positivo variando de 1 até k.

A metodologia descrita tem como idéia chave o conceito da assinatura única, chamada de seqüência UIO (*Unique Input/Output Sequence*), descrito no capítulo anterior, para cada estado da especificação do protocolo, por exemplo, a seqüência UIO para o estado 6 da figura 5.2 é *a/f c/d*. A escolha deste método de geração está pode ser justificada por que quase todas as FSM possuem este tipo de seqüência e o comprimento dessas seqüências é menor quando comparadas com os outros métodos de geração de seqüências de testes a partir de FSM's.

Para um estado s_i numa FSM que não possue uma seqüência UIO é usada uma assinatura que distingue s_i dos outros estados restantes. Este conceito está apresentado a seguir :

O processo de construção dessa assinatura é o seguinte : para cada estado s_j, pertencente ao conjunto de estados $\{s_1, s_2, ..., s_n\}$ com n elementos, existe uma seqüência de entrada/saída IO(i,j) (onde $i \neq j$) com comprimento inferior a n que pode distinguir s_i de s_j . A

primeira parte da assinatura para o estado s_i é uma seqüência de entrada $IO(i,1)$ que distingue s_i de s_1 e leva a máquina para o estado $s_{i,1}$. Então o menor caminho de $s_{i,1}$ para s_i é escolhido para levar a máquina de volta a s_i, este caminho é chamado de seqüência de transferência $T_i(1)$. O procedimento é aplicado de novo para distinguir s_i de todos os outros estados. A seguinte seqüência é usada como assinatura para os estados :

$$LD(s_i) = IO(i,1) \,@\, \overset{n}{\underset{k=2}{\#}} T_i(K-1) @ IO(i,k) \text{ onde } k \neq 1 \text{ para } i \neq 1$$

ou

$$LD(s_i) = IO(1,2) \,@\, \overset{n}{\underset{k=3}{\#}} T_1(K-1) @ IO(1,k) \text{ para } i = 1$$

Onde @ e # são usados para representar a concatenação de seqüências. O limite superior para o comprimento dessa assinatura é $2n^2$. Se algum estado não possuir uma seqüência UIO com comprimento inferior a $2n^2$ então a assinatura $LD(s_i)$ é usada.

O algoritmo para a geração das seqüências UIO executa os seguintes passos:

Passo (1) : para cada estado, são calculadas todas as seqüências de *entrada/saída* e é feita uma verificação para ver se cada uma dessas seqüências é única.

Passo (2) : se nenhuma seqüência única de comprimento 1 (um) for encontrada, então o mesmo procedimento é repetido para todas as seqüências de comprimento 2.

Passo (3) : repetir este procedimento para seqüências maiores até uma seqüência UIO ser encontrada ou o comprimento das seqüências exceder $2n^2$.

As seqüências de teste geradas pelo algoritmo descrito devem verificar se cada estado e cada ramo (ou transição de estado) da especificação existe na implementação e se cada um desses ramos possui um *label* (*i/o*) correto, para tal as seqüências de teste devem passar por todos os ramos. Com isso, para cada ramo a seqüência de teste executa os seguintes passos :

1. leva a implementação para o seu estado inicial;
2. aplica uma entrada para causar a transição de estado neste ramo;
3. aplica a seqüência UIO para o estado final.

Após ter visitado cada ramo, o teste deve levar a implementação para o seu estado inicial através da entrada de *reset* (*ri*).

O algoritmo para geração de seqüências de teste executa os seguintes passos :

Passo (1) : a partir do estado inicial calcular os menores caminhos para todos os estados usando o *Dijkstra's Algorithm* **[Gondran 85]**. A descrição dos passos deste algoritmo é encontrada no anexo I.

Passo (2) : gerar um roteiro de todos os ramos. O roteiro de ramo *e* exige a seguinte seqüência de teste, chamada de sub-seqüência :

$$TE(\mathrm{e}) = P(Head(e)) \,@\, Label(e) \,@\, UIO(Tail(e))$$

O roteiro de todos os ramos exige a seguinte seqüência de teste :

$$TS = \mathop{\#}_{e \in E} ri / null @ TE(e)$$

onde $P(s)$ é a transição do menor caminho entre o estado s e o estado inicial

5.3 - Uma Ferramenta para a Geração de Seqüências de Teste

Esta ferramenta foi implementada usando a linguagem C rodando em estações de trabalho SUN® SPARC em conjunção com o sistema operacional UNIX® e é constituída de duas partes, a primeira parte executa o passo 3 (transformação da máquina de estados) da metodologia descrita acima, a segunda parte é a ferramenta para geração de seqüência de teste propriamente dita, esta parte da ferramenta implementa o método proposto em **[Sabnani 88]**.

Para executar a primeira parte da ferramenta o usuário deve executar o programa ***transf*** e fornecer o *nome* do arquivo gerado pelo ESTIM (extensão *.proj.lab.l.auto*), o resultado são dois arquivos todos com o mesmo *nome* do arquivo gerado pelo ESTIM, porém, um com a extensão *.tim* (FSM intermediária) e o outro com a extensão *.fsm* (FSM de entrada para a ferramenta de geração de seqüências de teste).

Para executar a segunda parte da ferramenta o usuário deve executar o programa ***seqt*** e fornecer o *nome* do arquivo contendo a FSM (extensão *.fsm*), o resultado é um arquivo com o mesmo nome do arquivo que contém a FSM com a extensão *.tst* onde estão contidas as seqüências de teste para a FSM.

Nesta parte são dadas opções saída para que o usuário possa visualizar na tela o resultado da execução dos algoritmos implementados (algoritmo do caminho mais curto, algoritmo de seqüências UIO e algoritmo das seqüências de teste), e com isso verificar todo o processo de construção das seqüências de teste. A figura 5.3 ilustra a tela de apresentação da ferramenta.

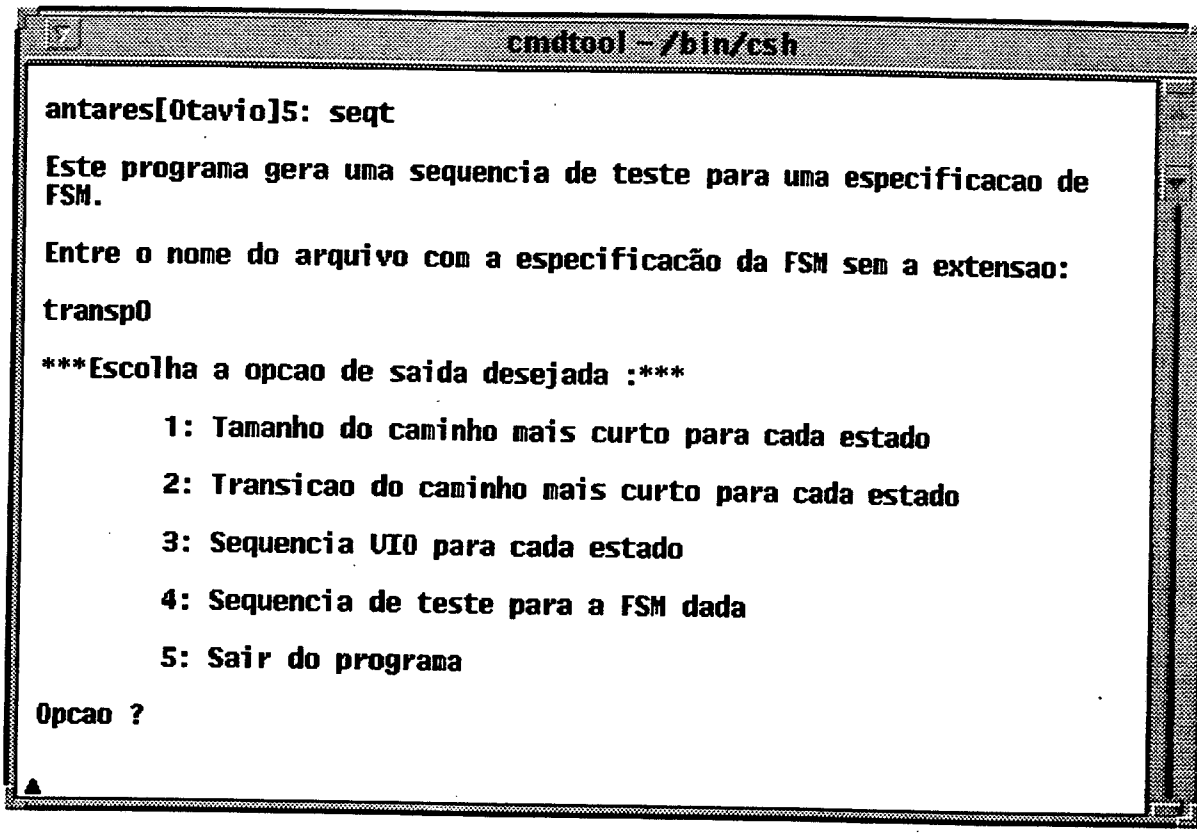

Figura 5.3 - Tela de apresentação da ferramenta implementada

A figura 5.3 ilustra a metodologia proposta. Deve ser notado que a transformação da especificação (ESTELLE → ESTELLE*) só é necessária quando a especificação do protocolo de comunicação já se encontrar descrita em ESTELLE, caso o protocolo se encontre especificado informalmente ele pode ser especificado diretamente em ESTELLE* não necessitando passar por esta etapa.

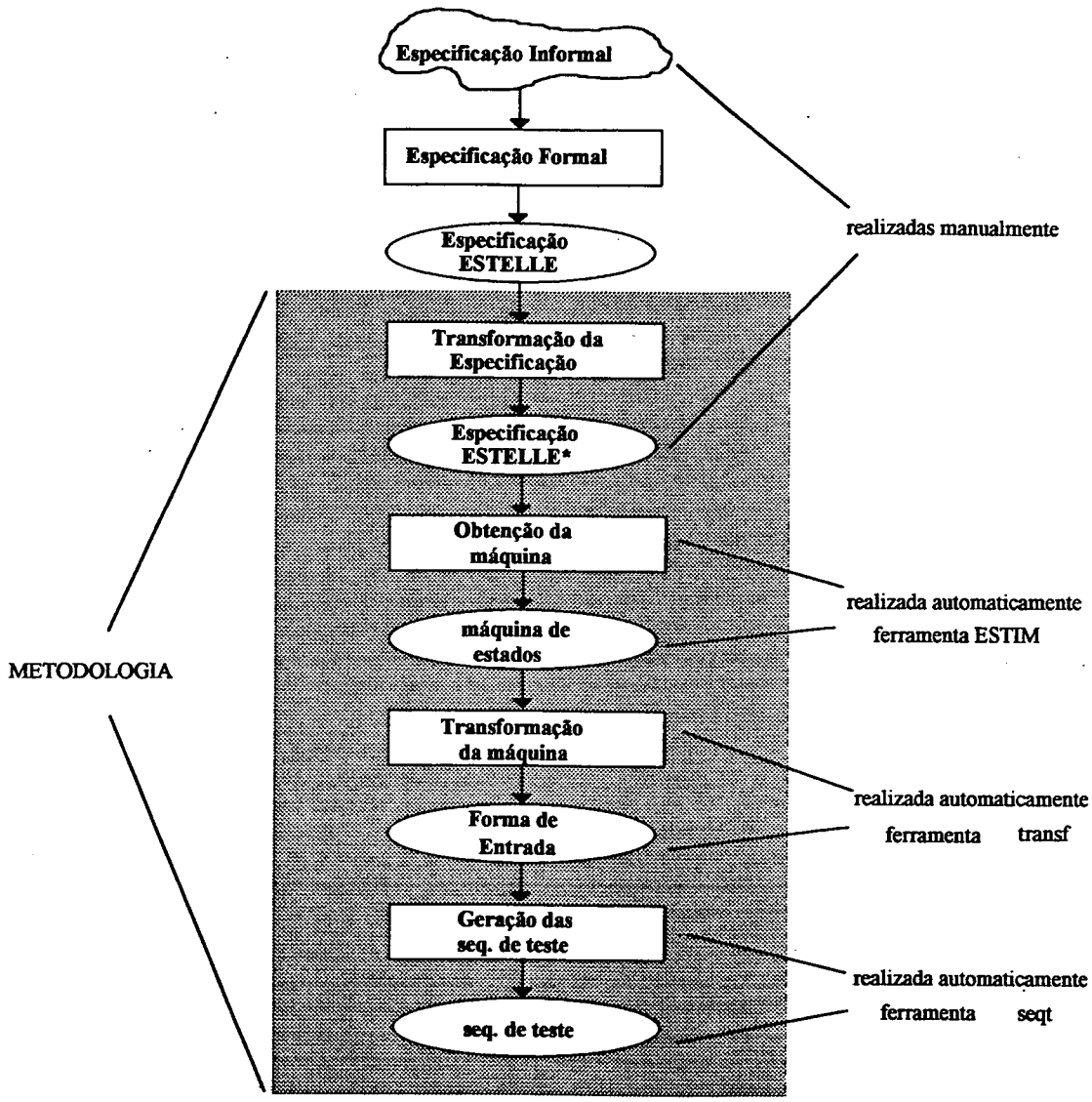

Figura 5.4 - Metodologia proposta

5.4 - Exemplos de Aplicação

Com o objetivo de ilustrar e validar o uso da metodologia proposta, tanto a parte de transformação quanto a parte de geração de seqüências de teste, é apresentada a seguir, a sua aplicação a dois exemplos encontrados na literatura.

O primeiro exemplo está especificado diretamente em FSM e permite ilustrar a aplicação direta do método UIO, sendo apresentados, inclusive, resultados intermediários do método.

O segundo exemplo, mais completo, foi especificado em ESTELLE* e permite mostrar todas as etapas que foram propostas na metodologia. Os resultados obtidos foram expostos nos anexos apresentados ao final do documento.

5.4.1 - Fase de Transporte de Dados do Protocolo Bit-Alternante

Para este exemplo foi utilizado a seção de transporte do protocolo bit alternante que encontra-se especificada como uma FSM em **[Sabnani 88]**, este exemplo foi escolhido devido este protocolo ser muito didático. Como este protocolo já se encontra na forma de uma FSM, figura 5.6, então foi montado o arquivo de entrada da ferramenta *seqt*, este arquivo está mostrado na figura 5.5, as seqüências de testes geradas pela aplicação dos algpritmos implementados na ferramenta seqt são mostradas na figura 5.7.

```
(0, tuser?data, null, 1)
(1, null, dmd!data0, 2)
(2, null, timer!start, 3)
(3, timer?out, null, 1)
(3, amd?ack0, null, 4)
(4, tuser?data, null, 5)
(5, null, dmd!data1, 6)
(6, null, timer!start, 7)
(7, amd?ack1, null, 0)
(7, timer?out, null, 5)
```

***Figura 5.5** - Arquivo de entrada da ferramenta seqt*

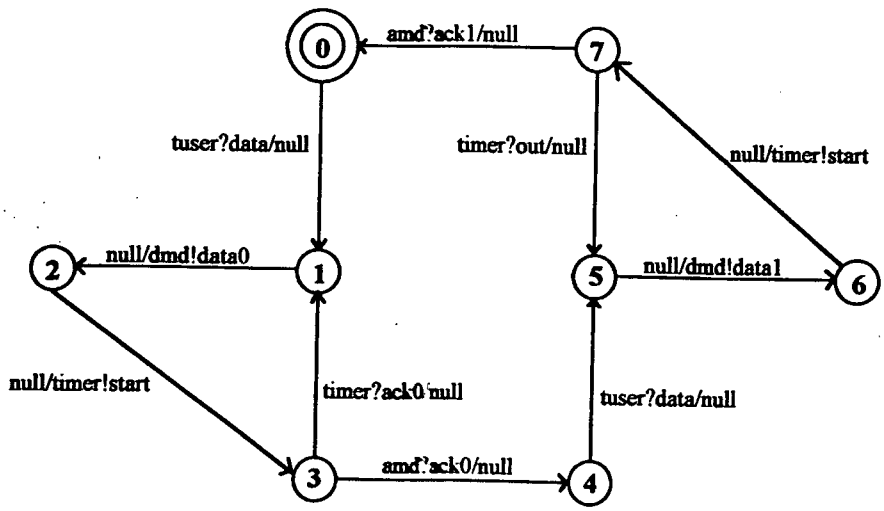

***Figura 5.6** - FSM do arquivo acima*

A tabela 5.1 apresenta o resultado da aplicação do algoritmo do caminho mais curto à FSM mostrada na figura 5.6.

Estado	Tamanho do caminho mais curto	Transição do caminho mais curto
0	0	ri/null
1	1	tuser?data/null
2	2	tuser?data/null null/dmd!data0
3	3	tuser?data/null null/dmd!data0 null!timer/start
4	4	tuser?data/null null/dmd!data0 null!timer/start amd?ack0/null
5	5	tuser?data/null null/dmd!data0 null!timer/start amd?ack0/null tuser?data/null
6	6	tuser?data/null null/dmd!data0 null!timer/start amd?ack0/null tuser?data/null null/dmd!data1
7	7	tuser?data/null null/dmd!data0 null!timer/start amd?ack0/null tuser?data/null null/dmd!data1 null!timer/start

Tabela 5.1 - Resultado do algoritmo do caminho mais curto

A tabela 5.2 apresenta o resultado da aplicação do algoritmo de geração de seqüências UIO para a FSM mostrada na figura 5.6.

Estado	Seqüência UIO
0	tuser?data/null null/dmd!data0
1	null/dma!data0
2	null/timer!start timer?out/null null/dmd!data0
3	amd?ack0/null
4	tuser?data/null null/dmd!data1
5	null/dmd!data1
6	null/timer!start amd?ack1/null
7	amd?ack1/null

***Tabela 5.2** - Resultado do algoritmo de geração de seqüências UIO*

Na figura 5.7 estão apresentadas as seqüências de teste geradas para a FSM da figura 5.6. Estas seqüências, assim como aquelas apresentadas no item anterior, são usadas para testar conformidade fraca, e estão armazenadas em um arquivo com a extensão *.tst*.

ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start timer?out/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null
ri/null tuser?data/null null/dmd!data0 null/timer!start timer?out/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1 null/timer!start amd?ack1/null
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1 null/timer!start amd?ack1/null
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1 null/timer!start amd?ack1/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1 null/timer!start timer?out/null null/dmd!data1
ri/null ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1 ri/null tuser?data/null null/dmd!data0
ri/null tuser?data/null null/dmd!data0 null/timer!start amd?ack0/null tuser?data/null null/dmd!data1 null/timer!start ri/null tuser?data/null null/dmd!data0

Figura 5.7 - Seqüências de Teste

Estes resultados correspondem aos resultados encontrados em **[Sabnani 88]** o que ajuda a validar a ferramenta implementada.

5.4.2 - Protocolo Transporte Simplificado

Para este exemplo o protocolo de transporte simplificado proposto em **[Sarikaya 89]** foi especificado em ESTELLE.

O protocolo de transporte simplificado é baseado nas seguintes primitivas de serviço :

- TCONreq - solicitação de estabelecimento de conexão de transporte
- TCONind - indicação de estabelecimento de conexão de transporte
- TCONresp - resposta a indicação de estabelecimento de conexão de transporte
- TCONconf - confirmação do estabelecimento de conexão de transporte
- TDATAreq - solicitação de estabelecimento de conexão de dados
- TDATAind - indicação de estabelecimento de conexão de dados
- TDISreq - solicitação de cancelamento da conexão de transporte
- TDISind - indicação de cancelamento da conexão de transporte

As unidades de dados de protocolos trocadas entre as entidades são as seguintes :

- CR - *Connect Request*
- CC1 - *Connect Confirm* (negativo)
- CC2 - *Connect Confirm* (positivo)
- DT, AK - *Data, Acknowledged*
- DR - *Disconnect Request*
- DC - *Disconnect Confirm*

Podemos descrever, informalmente, o funcionamento do protocolo, segundo as seguintes etapas :

I. ESTABELECIMENTO DA CONEXÃO DE TRANSPORTE

* ***Iniciada localmente :***

1. usuário emite a primitiva TCONreq
2. entidade envia PDU-CR à entidade par e aguarda confirmação
3. se a PDU recebida é positiva (CC2), a entidade encaminha confirmação ao usuário e a conexão é considerada aberta (OPEN)
4. se a PDU é negativa (CC1), a entidade encaminha uma primitiva de serviço TDISind ao usuário e envia a PDU-DR à entidade par. A conexão é considerada fechada (CLOSED).

* ***Iniciada remotamente (pela entidade par) :***

1. entidade recebe PDU-CR e emite primitiva TCONind ao usuário, ficando a espera de resposta
2. usuário emite resposta positiva (primitiva TCONresp), entidade emite PDU-CC2 considerando a conexão estabelecida (OPEN)
3. usuário emite resposta negativa (primitiva TDISreq), entidade emite PDU-DR e aguarda resposta. Em seguida, a entidade encaminha uma primitiva de serviço TDISconf ao usuário e envia a PDU-DR à entidade par. A conexão é considerada fechada (CLOSED).

II. FASE DE TRANSFERÊNCIA DE DADOS :

* ***emissão do dado***

1. entidade recebe primitiva TDATAreq e emite o dado; utilizando PDU-DT
2. entidade recebe reconhecimento do dado; utilizando PDU-AK

* ***recepção do dado***

1. entidade recebe PDU-DT
2. entidade emite dado ao usuário via primitiva TDATAind
3. entidade retorna reconhecimento à entidade par

III. TERMINAÇÃO DA CONEXÃO DE TRANSPORTE

* ***iniciada localmente (usuário)***

1. entidade recebe pedido de desconexão do usuário (TDISreq)
2. entidade emite PDU-DR e fica a espera de resposta (da entidade par)
3. recebida a PDU-DC, entidade envia confirmação de desconexão (primitiva TDISconf) e retorna ao estado inicial (CLOSED)

* ***iniciada localmente (pela entidade de transporte)***

1. entidade emite indicação de terminação (TDISind) e PDU-DR, ficando a espera de resposta
2. recebida a PDU-DC, entidade envia confirmação de desconexão (primitiva TDISconf) e retorna ao estado inicial (CLOSED)
3. a entidade encaminha uma primitiva de serviço TDISconf ao usuário e envia a PDU-DR à entidade par. A conexão é considerada fechada (CLOSED).

* ***iniciada remotamente (pela entidade par)***

1. entidade recebe PDU-DR
2. envia indicação de desconexão (TDISind) e resposta PDU-DC

A especificação ESTELLE* do protocolo está apresentada no anexo II.

Como se pode verificar nesta especificação, a arquitetura especificada corresponde àquela mostrada na figura 5.1, sendo composta dos módulos USER_A, T_ENT_A e NETWORK que representam, respectivamente, o usuário da camada de transporte,a entidade de transporte e o serviço de comunicação.

O módulo USER_A comunica-se com o módulo T_ENT_A através do canal CH1 que interliga os pontos de interação denominados TSAP (tanto no módulo USER_A quanto em T_ENT_A). As interações que podem ser trocadas entre estes módulos são : TCONreq, TCONresp, TDISreq, TDISresp, TDATAreq e NULL, num sentido e TCONind, TCONconf, TDISind TDISconf e TDATAind, no outro.

O canal CH2, que interliga os módulos NETWORK e T_ENT_A através dos pontos de interação NSAP, permitindo representar a troca de PDU's entre as entidades de transporte. As interações circulando neste canal são : CR, CC, DR, DC, DT, CC1 e CC2 em ambos os sentidos.

Conforme descrito na seção 5.2.1, os corpos de módulos de USER_A e NETWORK são definidos de modo a representar as comunicações externas nas interfaces superior e inferior do módulo T_ENT_A. O corpo deste, por sua vez, descreve o comportamento do protocolo de transporte como apresentado na especificação informal.

A próxima etapa será obter o autômato que descreve o comportamento observável do protocolo de transporte.

Para isto, ativa-se a ferramenta ESTIM, indicando o nome do arquivo que contém a especificação ESTELLE*. Terminada a inicialização, deve ser ativada a facilidade de geração do grafo de alcançabilidade (através do comando *"graph"*).

Em seguida, para que se possa lançar a geração do grafo, é necessário informar as fronteiras que servirão de referência para distinguir eventos observáveis e eventos invisíveis - é a definição de partição. A escolha é feita de modo que apenas o módulo T_ENT_A permaneça no interior da partição, os demais módulos vindo fazer parte do ambiente. A figura 5.8 ilustra esta escolha.

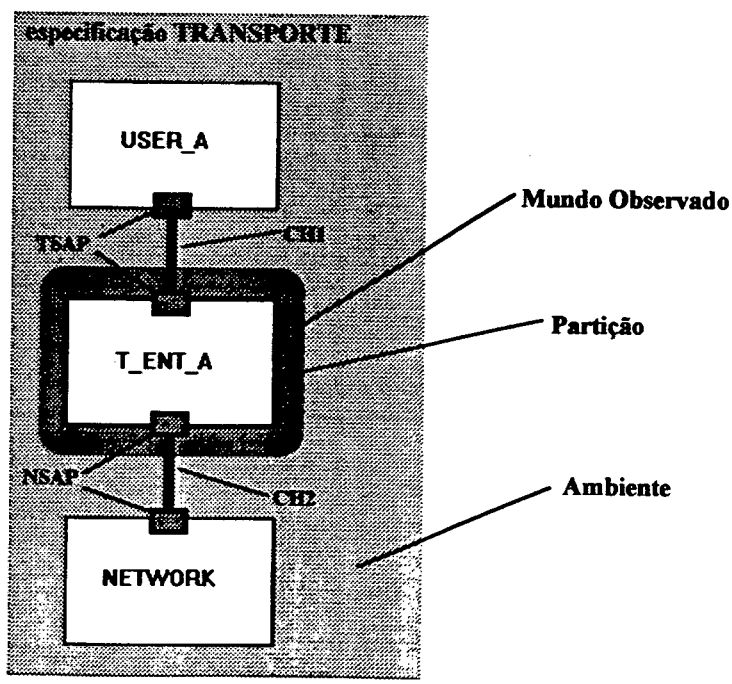

Figura 5.8 - Forma geral da especificação Transporte

Sendo assim, a ferramenta vai considerar como observáveis, todos os eventos que estejam associados a trocas de interações nos canais CH1 e CH2. Qualquer outro evento será considerado invisível e será, portanto, etiquetado por *"nil"*.

Lançada a geração de grafo, o resultado obtido deverá sofrer ainda o processo de redução por equivalência de rastro. Isto é feito invocando-se o comando *"proj -l"* de dentro do ambiente ESTIM.

O autômato obtido representa, finalmente, o comportamento observável da entidade de transporte, sobre o qual serão aplicados as demais etapas da metodologia proposta. O arquivo obtido é apresentado no anexo III.

O resultado foi aplicado a ferramenta *transf* para se obter a forma da FSM de entrada da ferramenta *seqt* (geração de seqüências de teste). Esta FSM foi aplicada para a ferramenta *seqt* de onde se obteve os resultados mostrados abaixo. Para facilitar a visualização dos resultados foi montada a FSM, mostrada na figura 5.10, a partir do arquivo de entrada da ferramenta *seqt*, mostrado na figura 5.9.

```
(1, CR, TCONIND, 5)
(1, TCONREQ, CR, 4)
(4, CC1, TDISIND-DR, 14)
(4, CC2, TCONCONF, 12)
(5, TCONRSP, CC, 12)
(5, TDISREQ, DR, 7)
(7, nil, TDISCONF, 1)
(12, DR, DC-TDISIND, 7)
(12, DT, TDATAIND, 12)
(12, TDATAREQ, DT, 12)
(12, TDISREQ, DR, 14)
(12, nil, TDISIND-DR, 14)
(14, DC, TDISCONF, 1)
```

Figura 5.9 - Arquivo de entrada da ferramenta seqt

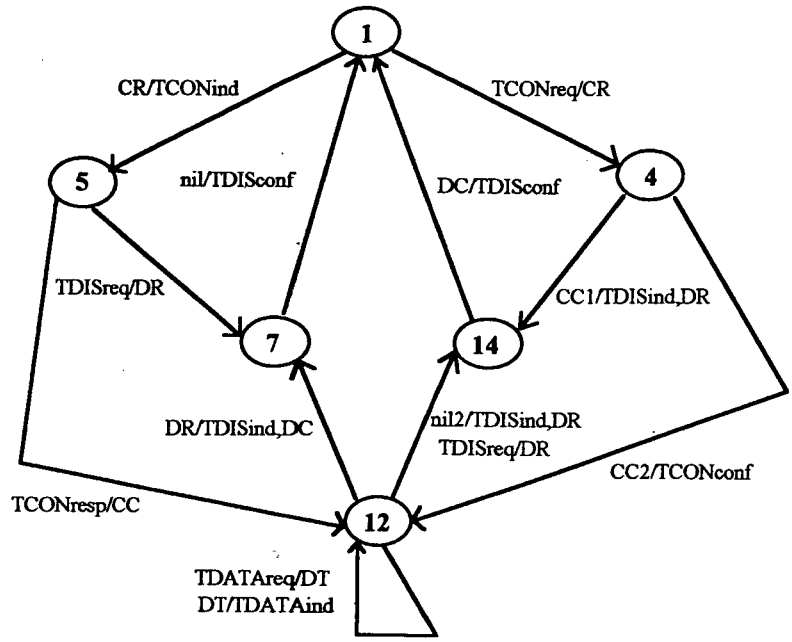

***Figura 5.10** - FSM do arquivo acima*

Abaixo será dada a equivalência entre o número do estado e o seu significado :

Estado	Significado
1	closed
4	wait_for_CC
5	wait_for_TCONresp
7	closing
12	open
14	wait_for_DC

***Tabela 5.3** - Equivalência dos estados*

A tabela 5.4 apresenta o resultado da aplicação do algoritmo do caminho mais curto à FSM mostrada acima.

Estado	Tamanho do caminho mais curto	Transição do caminho mais curto
1	0	ri/null
4	1	TCONREQ/CR
5	1	CR/TCONIND
7	2	CR/TCONIND TDISREQ/DR
12	2	TCONREQ/CR CC2/TCONCONF
14	2	TCONREQ/CR CC1/TDISIND-DR

Tabela 5.4 - Resultado do algoritmo do caminho mais curto

A tabela 5.5 apresenta o resultado da aplicação do algoritmo de geração de seqüências UIO para a FSM mostrada na figura 5.10.

Estado	Seqüência UIO
1	CR/TCONIND
4	CC1/TDISIND-DR
5	TCONRSP/CC
7	NULL/TDISCONF
12	DR/DC-TDISIND
14	DC/TDISCONF

Tabela 5.5 - Resultado do algoritmo de geração de seqüências UIO

Na figura 5.11 estão apresentadas as seqüências de teste geradas para a FSM da figura 5.10. Estas seqüências são usadas para testar conformidade fraca. Como foi dito na seção 5.4 estas seqüências estão armazenadas em um arquivo com a extensão *.tst*.

ri/null CR/TCONIND TCONRSP/CC
ri/null TCONREQ/CR CC1/TDISIND-DR
ri/null TCONREQ/CR CC1/TDISIND-DR DC/TDISCONF
ri/null TCONREQ/CR CC2/TCONCONF DR/DC-TDISIND
ri/null CR/TCONIND TCONRSP/CC DR/DC-TDISIND
ri/null CR/TCONIND TDISREQ/DR NULL/TDISCONF
ri/null CR/TCONIND TDISREQ/DR NULL/TDISCONF CR/TCONIND
ri/null TCONREQ/CR CC2/TCONCONF DR/DC-TDISIND
ri/null TCONREQ/CR CC2/TCONCONF DT/TDATAIND DR/DC-TDISIND
ri/null TCONREQ/CR CC2/TCONCONF NULL/TDISIND-DR DC/TDISCONF
ri/null TCONREQ/CR CC2/TCONCONF TDATAREQ/DT DR/DC-TDISIND
ri/null TCONREQ/CR CC2/TCONCONF TDISREQ/DR DC/TDISCONF
ri/null TCONREQ/CR CC1/TDISIND-DR DC/TDISCONF CR/TCONIND
ri/null ri/null CR/TCONIND
ri/null TCONREQ/CR ri/null CR/TCONIND
ri/null CR/TCONIND ri/null CR/TCONIND
ri/null CR/TCONIND TDISREQ/DR ri/null CR/TCONIND
ri/null TCONREQ/CR CC2/TCONCONF ri/null CR/TCONIND
ri/null TCONREQ/CR CC1/TDISIND-DR ri/null CR/TCONIND

***Figura 5.11** - Seqüências de teste*

Um ponto importante a ressaltar sobre o resultado da aplicação do algoritmo de geração de seqüências UIO é que para FSM's que possuam estados com mais de uma seqüência UIO de tamanho mínimo o algoritmo adota a primeira dessas seqüências que ele encontrar.

Para cada um dos arquivos gerados pela ferramenta foi criado um ícone especial, o que ajuda na visualização destes quando dispostos no *filemanager* da estação de trabalho SUN® SPARC. Um exemplo deste *filemanager* está apresentado na figura 5.12.

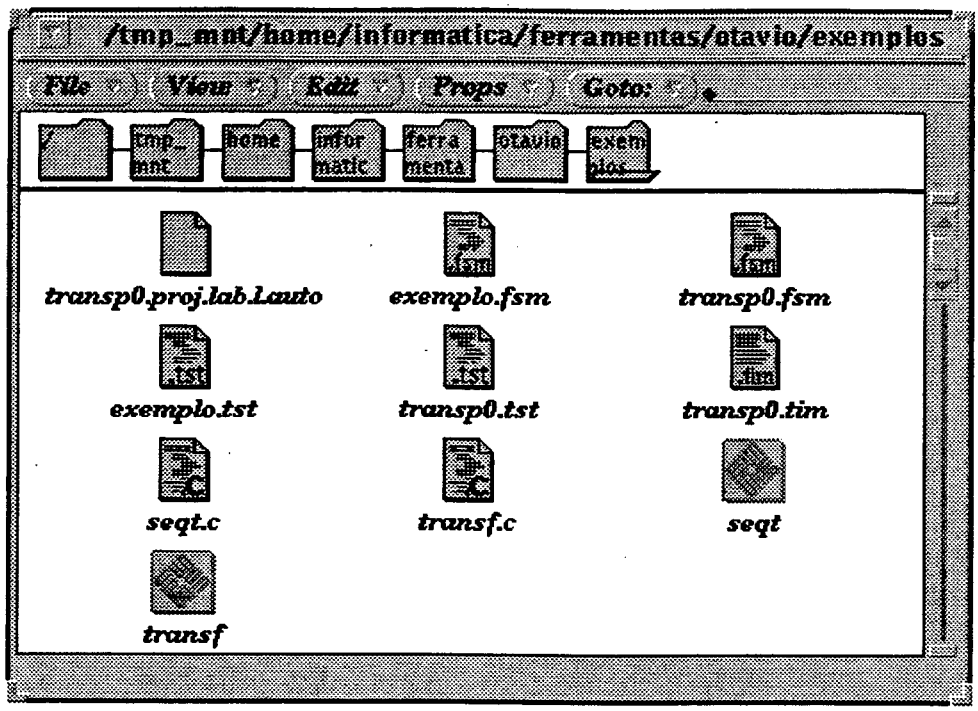

Figura 5.10 - Filemanager contendo os arquivos gerados e usados pela ferramenta

5.5 - Restrições ao Uso da Ferramenta de Geração de Seqüência de Teste

Existem duas restrições que devem ser observadas para o correto manuseio da ferramenta implementada, uma para o uso da ferramenta para a geração de seqüências de teste a partir da técnica ESTELLE e outra para o uso da ferramenta para a geração de seqüências de teste a partir de uma FSM.

A restrição associada a técnica ESTELLE é a seguinte : caso exista na especificação original (sem os estados artificiais) um estado com uma interação (tanto de entrada como de saída) *null*, esta interação deve ser declarada como NULL, isto é necessário para que a transformação para a forma de entrada da ferramenta de geração seja possível de ser realizada.

A restrição para o uso de FSM é que o arquivo contendo a FSM que irá ser testada deve apresentar o formato do arquivo *.fsm* mostrado abaixo :

(*est_inicial*, *int_entrada*, *int_saída*, *est_final*)

onde : *est_inicial* é o estado de início da transição

int_entrada é a interação de entrada

int_saída é a interação de saída

est_final é o estado final da transição

5.6 - Conclusões

Neste capítulo foi proposta uma metodologia para a geração de seqüências de teste baseada na técnica ESTELLE, com a ajuda da ferramenta ESTIM, utilizando um método de geração baseado em FSM's.

O que se tentou mostrar, com a metodologia proposta, foi que utilizando uma ferramenta de suporte à técnica ESTELLE existente consegue-se obter seqüências de teste a partir de um método de geração baseado em FSM, ou seja, realizou-se uma integração entre uma ferramenta já existente com a ferramenta desenvolvida.

É lógico que existem um grupo de restrições que devem ser aplicadas à especificação, tanto em ESTELLE quanto na FSM, porém estas restrições visam reduzir o poder de expressão destas técnicas, para com isso resolver os problemas associados a testabilidade (não-determinismo e coordenação), por este motivo elas são necessárias para que se possa aplicar o método de geração escolhido baseado em FSM's e que aplica algumas restrições quanto ao modelo de FSM utilizado para a geração das seqüências de teste.

Outra questão abordada por este capítulo foi a apresentação da ferramenta implementada, explicando o seu funcionamento e a maneira como os resultados são apresentados para o usuário.

Com relação à aplicação da metodologia apresentada para FSM's grandes (FSM's com um grande número de estados) existem dois pontos a serem considerados; primeiro seria a geração do grafo de alcançabilidade pela ferramenta ESTIM e segundo seria com relação ao número e o comprimento das seqüências de testes geradas

Para o primeiro ponto pode-se dizer que a geração do grafo de alcançabilidade para FSM's grandes seria bastante demorada o que acarretaria na ocupação dos recursos computacionais durante um bom periodo de tempo, além do que poderia haver uma explosão combinatória dos estados do grafo, fazendo com que este grafo não consiga convergir para um determinado ponto.

Para o segundo ponto nota-se que o número e o comprimento das seqüências de testes geradas aumenta com o número de estados da FSM, o que resultaria em um número grande de seqüências bastante longas, isto ocorre devido ao algoritmo de geração de seqüências de testes utilizado.

Uma outra questão que não foi abordada neste trabalho seria com relação ao grau de cobertura das seqüências de testes geradas. Deve ser verificado se as seqüências de teste garantem que a implementação está *conforme* à especificação apenas passando um certo número de vezes em cada estado ou em cada transição de estado.

É importante ressaltar que a equivalência de rastros é a forma de equivalência mais simples que existe o que reduz o tempo de processamento na realização desta etapa. Este tipo de equivalência retira o não-determinismo, mas esconde *deadlocks* na especificação, porém é importante ressaltar que o objetivo dos testes de conformidade é dizer se a implementação está conforme a especificação e não dizer se a especificação do protocolo está correta, por esta razão é necessário que a especificação do protocolo seja validada antes de se gerar as seqüências de teste.

Está claro que a transformação de ESTELLE para ESTELLE* é uma simplificação na especificação, por exemplo o mecanismo de comunicação *rendez-vous* de ESTELLE* limita as filas FIFO de ESTELLE, porém estas simplificações são necessárias para que a geração do grafo de alcançabilidade seja limitada. O que pode ser notado em relação a estas simplificações é que elas são bastante simples quando comparadas com as simplificações propostas pelos outros métodos de geração de seqüências de testes a partir de ESTELLE, mostrado que a metodologia proposta simplifica este processo.

CAPÍTULO 6

CONCLUSÕES E PERSPECTIVAS

O estudo de teste de um sistema distribuído em geral, e de protocolos de comunicação em particular, utilizando algum tipo de modelo é uma área de pesquisa relativamente nova. Dentro desta idéia, este trabalho procurou descrever os conceitos importantes relacionados ao teste de protocolos de comunicação, especificamente aqueles relacionados ao teste de conformidade. Uma das etapas deste trabalho foi realizar uma extensa revisão bibliográfica visando detectar os principais tópicos relacionados com este tema, reunindo estes conceitos em um único documento. Com isso procurou-se desenvolver o texto deste trabalho de uma maneira clara, concisa e completa fazendo com que ele possa servir como uma importante fonte de consulta para trabalhos futuros nesta linha de pesquisa.

Dentro dos conceitos estudados foi dada uma atenção maior para a questão da geração de seqüências de testes a partir de especificações formais, visto que uma das principais contribuições deste trabalho foi verificar a possibilidade de se aplicar um método de geração de seqüências de teste baseado em FSM's para a técnica de descrição formal ESTELLE, utilizando a ferramenta ESTIM, disponível no LCMI (Laboratório de Controle e Microinformática), e com isso promover uma certa integração entre uma ferramenta já disponível e a ferramenta desenvolvida ao longo do trabalho.

A metodologia prosposta neste trabalho diminui consideravelmente o esforço que precisa ser realizado para a geração de seqüências de teste a partir da técnica ESTELLE, visto que as únicas restrições que devem ser respeitadas são àquelas referentes a versão estendida ESTELLE*, tais como o mecanismo de comunicação *rendez-vous*. Se estas restrições forem respeitadas no momento da especificação do protocolo, consegue-se obter seqüências de teste para esta especificação. Estas restrições são bastante simples quando comparadas àquelas impostas por outros métodos estudados na literatura.

A grande vantagem da metodologia é que em nenhum momento a especificação do protocolo passa por uma transformação pra algum tipo de modelo, por exemplo o modelo *Chart* e o modelo TOF apresentados no capítulo 4 para a geração de seqüências de testes a partir de LOTOS e ESTELLE respectivamente, que seja adequado para a geração de seqüências de teste, modelos estes que tiram uma das maiores vantagens das técnicas de descrição formal, o seu alto poder de expressão.

Está claro que a metodologia proposta necessita passar por uma série de testes mais rigorosos para que se possa comprovar a sua validade, com isso uma perspectiva para um trabalho futuro seria validar a ferramenta desenvolvida com especificações de protocolos mais complexas onde se possa observar uma série de situações que possam validar as hipóteses colocadas na metodologia proposta neste trabalho.

Uma outra questão que pode ser avaliada é a necessidade de um estudo aprofundado relativo à aplicação das seqüências de testes obtidas em cima da implementação do protocolo. Este estudo deve levar em consideração de que maneira estas seqüências devem ser aplicadas à implementação e como os resultados obtidos após a aplicação destas seqüências devem ser avaliados para que se possa garantir que a implementação está realmente *conforme* com a especificação do protocolo.

A proposta de um ambiente de auxílio ao teste que permita derivar seqüências de teste a partir de especificações formais também pode ser vista como uma perspectiva de trabalhos futuros, visto que, na bibliografia relacionada encontra-se um conjunto de ambientes relativamente extenso, implementando os mais diversos métodos, entretanto estes ambientes são concebidos normalmente considerando um único método, sendo orientado a especificações formais representadas num modelo bem definido, podendo este ser máquinas de estado finito ou uma das técnicas de descrição formal existente. Por outro lado, é de nosso conhecimento que especificações formais dos inúmeros protocolos de comunicação existentes não estão representadas, necessariamente segundo uma mesma técnica.

Por esta razão, esta proposta vai na direção de um ambiente de auxílio ao teste de protocolos de comunicação que permita obter seqüências de teste de protocolos de comunicação a partir de especificações formais representadas segundo diferentes técnicas de descrição formal.

O ambiente proposto, cujo diagrama em blocos é mostrado na figura abaixo, é composto de um conjunto de módulos de geração de seqüências de teste. Cada um destes módulos será responsável da implementação de um método de geração de seqüências de teste para uma técnica em particular.

Para este ambiente ser possível basta ser escolhido e implementado um dos algoritmos de geração de seqüências de testes baseado na técnica LOTOS, apresentados no capítulo 4 deste trabalho.

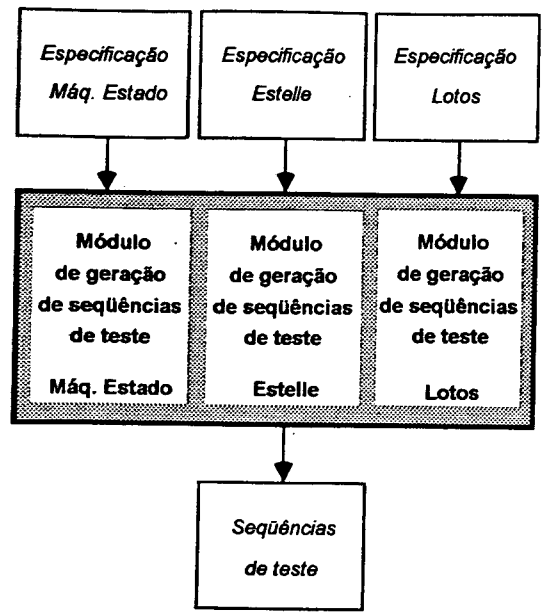

***Figura 6.1** - Diagrama em blocos do ambiente proposto.*

REFERÊNCIAS BIBLIOGRÁFICAS

[Arakawa 91] ARAKAWA, N., SOCONKA, T. *"A test case generation method for concurrent programs"*. Fourth Int'l Workshop on Protocol Test Systems, Amsterdam, 1991. North-Holland.

[Beizer 90] BEIZER B. *"Software Testing Techniques"*, Van Nostrand Reinhold, New York, NY, USA, second edition, 1990.

[Bosik 91] BOSIK, B.S., UYAR, M. Ü. *"Finite state machine based formal methods in protocol conformance testing : from theory to implementation"*, Computer Network and ISDN Systems 22 (1991), pp. 7-33.

[Budkowski 87] BUDKOWSKI S., DEMBINSKI P. *"An Introduction to ESTELLE : A Specification Language for Distributed Systems"*, Computer Networks and ISDN Systems, Vol. 14, 1987, pp. 3-23.

[CCITT/Z-100] CCITT/SGXI/WP3-1 *"SDL Specification and Description Language. CCITT Recommendations Z.100-Z104"*, 1988.

[Chow 78] CHOW, Tsun S. *"Testing Software Design Modeled by Finite State Machines"*. IEEE Transactions on *Software* Engineering Vol. SE-4 n° 3. Maio 1978. 178-187.

[Chun 90] CHUN, Woojik, AMER, Paul D. *"Test Case Generation for Protocols Specified in ESTELLE"*. Proceedings of the IFIP International Conference on Formal Description Techniques - FORTE'90. Madrid (Espanha), Outubro (1990). 197-209.

[Courtiat 87a] COURTIAT J. P. *"Contribuition à la Description Formelle de Protocoles"* Thèse de Doctorat d'Etat. Université Paul Sabatier, Toulose, 1987.

[Courtiat 87b] COURTIAT J. P, et alli. *"ESTELLE* : An ISO language for distributed algorithms and protocols"*. Technology and Science of Informatics. vol. 6, n° 5, pp 311-324, 1987.

[Danthine 80] DANTHINE, A. A. S. *"Protocol representation with Finite-State Models"*, IEEE Transactions on Communications, Vol. COM-28, n. 4, April (1980), pp. 632-643.

[Diaz 82] DIAZ, M. "Modeling and Analysis of Communication and Coopereation Protocols using Petri Net Based Models", Computer Networks & ISDN Systems, Vol. 6 (1982), pp. 419-441.

[Diaz 89] DIAZ M., DUFAU J., GROZ R. "Experiences using ESTELLE within Sedos ESTELLE Demonstrator", Proceedings of the Second International Conference on Formal Description Techniques FORTE'89, Vancouver, Canada, December 1989, Ed. Noth-Holland, pp. 455-470.

[Favreau 87] FAVREAU, Jean-Philipe, LINN, Richard J. "Automatic Generation os Test Scenario Skeletons from Protocol Specifications Written in ESTELLE". IFIP 1987. 191-201

[Forghani 90] FORGHANI, Behdad, SARIKAYA, Behçet. "Automatic Dynamic Behaviour Generation in TTCN Format from ESTELLE Specifications". Proceedings of X IFIP International Conference on Protocol Specification, Testing and Verification (1990). 125-139.

[Fujiwara 91] FUJIWARA, S., BOCHMANN, G. von. "Testing non-deterministic state machines with fault coverage". Fourth Int'l Workshop on Protocol Test Systems, Amsterdam, 1991. North-Holland.

[Ghedamsi 92] GHEDAMSI, A. , DSSOULI, R., BOCHMANN, G. von. "Diagnostic test for single transition faults in non-determinism finite state machines". Fifth Int'l Workshop on Protocol Test Systems, Amsterdam, 1992. North-Holland.

[Gondran 85] GONDRAN, M., MINOUX, M. "Graphes et Algorithmes". Collection de la Direction des Études et Recherches d'Electricité de France. Editions Eyrolles, 1985.

[Goscinski 91] GOSCINSKI A. "Distributed Operating Systems : The Logical Design", Addison-Wesley, Reading, MA, USA, 1991.

[Gottfried 93] GOTTFRIED B. S., "Programando em C", 1ª. edição, São Paulo - SP, Editora Makron Books, 1993.

[Gouda 91] GOUDA M. G., MULTARI N. J. "Stabilizing communication protocols", IEEE Transactions on Computers, 40(4), pp. 448-458, Abril 1991.

[ISO8807] ISO IS 8807. LOTOS, *"A formal Description Technique Based on the Temporal Ordering of Observational Behaviour"*, Novembro 1988.

[ISO9074] ISO IS 9074. ESTELLE - *"A Formal Description Technique Based on a Extended State Transition Model"*, International Standardization Organization, Novembro (1988).

[ISO9646-1] ISO/IEC JCT 1/ SC 21. *"Information Processing Systems - OSI Conformance Testing Methodology and Framework, Parts 1"*, Novembro (1988).

[ISO9646-2] ISO/IEC JCT 1/ SC 21. *"Information Processing Systems - OSI Conformance Testing Methodology and Framework, Parts 2"*, Novembro (1988).

[ISO9646-3] ISO/IEC JCT 1/ SC 21. *"Information Processing Systems - OSI Conformance Testing Methodology and Framework, Part 3"*, February (1989).

[Lee 92] LEE Do Y., LEE Jai Y. *"Test Generation for the Specification Written in ESTELLE"*. Pohang, Kyungbuk, KOREA 790-600. 294-311.

[Leite 92] LEITE M. M. *"Uma Ferramenta para Especificação de Protocolos dentro do Contexto ESTELLE : Um Tradutor ASN.1/ESTELLE"*, Dissertação de Mestrado, CPGEEL-UFSC, Florianópolis-SC, abril de 1992.

[Lin 89] LIN H. P., STOVALL H. E. *"Self-synchronizing communication protocols"*, IEEE Transactions on Computers, 38(5), pp. 609-625, Maio 1989.

[Linn 90] LINN R. J. *"Conformance Testing for OSI Protocols"*. Computer Networks and ISDN Systems 18 (1989/90) 203-219.

[Liskov 86] LISKOV B., GUTTAG, J. *"Abstraction and Specification in Program Development"*, MIT Press, 1986.

[Lloret 90] LLORET J. C. *"Réseaux Predicat/Transition Etiquétés pour la Modélisation et la Vérification de Systèmes Informatiques Répartis"* Thèse de Doctorat de l'Université Paul Sabatier, Toulose, 1990.

[Manhas 94] MANHAS JR, E. B. *"Uma Metodologia para o Desenvolvimento de Aplicações Distribuídas baseada na Técnica de Descrição Formal*

ESTELLE", Dissertação de Mestrado, CPGEEL-UFSC, Florianópolis-SC, julho de 1994.

[Miller 87] MILLER R. E. *"The construction of self-synchronizing finite state protocols"*, Distribuited Computing, 2(2) pp. 104-112, 1987.

[Naik 92] NAIK K., SARIKAYA B. *"Testing Communication Protocols"*, IEEE *Software* (Janeiro 1992) 27-36.

[Phalippou 90] PHALIPPOU, Marc, GROZ, Roland. *"From ESTELLE Specifications to Industrial Test Suites, using an Empirical Approach"*. Proceedings of the IFIP International Conference on Formal Description Techniques - FORTE'90. Madrid (Espanha), Outubro (1990). 179-196.

[Pitt 90] PITT, David H., FREESTONE David. *"The Derivation of Conformance Tests from LOTOS Specifications"*. IEEE Transactions on *Software* Engeneering Vol. 16 n° 12 (Dezembro 1990). 1337-1343.

[Rayner 87] RAYNER, D. *"OSI Conformance Testing"*. Computer Networks and ISDN Systems 14 (1987) 79-98.

[Sabnani 88] SABNANI, Krishan, DAHBURA, Anton. *"A Protocol Test Generation Procedure"*. Computer Networks and ISDN Systems 15 (1988). 285-297.

[Saqui 90] SAQUI-SANNES, P. de. *"Prototypage d'un Environnement de Validation de Protocoles : Application à l'Approche ESTELLE"*, Tese de Doutorado, Toulouse, Abril, 1990.

[Sarikaya 89] SARIKAYA, Behçet. *"Conformance Test: Architectures and Test Sequences"*. Computer Networks and ISDN Systems 17 (1989). 111-126.

[Sidhu 89] SIDHU, Deepinder P., LEUNG, Ting-kau. *"Formal methods for protocol testing : A detailed study"*. IEEE Transactions on *Software* Engineering, 15 n°4, Abril 1989, pp. 413-426.

[Silveira 91] SILVEIRA J. L. *"Estudo da Utilização do padrão MMS em Aplicações Fabris"*, Dissertação de Mestrado, CPGEEL-UFSC, Florianópolis-SC, dezembro de 1991.

[Tripathy 91] TRIPATHY, Piyu, SARIKAYA, Behçet. *"Test generation from LOTOS Specifications"*. IEEE Transactions on Computer Vol. 40 n° 4 (Abril 1991). 543-552.

[Ural 87] URAL, Hasan. *"A Test Derivation Method for Protocol Conformance Testing"*. IFIP 1987. 347-358.

[Vuong 93] VUONG, S.T., LOUREIRO, A.A.F., CHANSON, S.T. *"A Framework for the Design for Testability of Comunications Protocols"*. 11° Simpósio Brasileiro de Redes de Computadores , Maio 1993.

[West 78] WEST C.H., *"General technique for communications protocol validation"*. IBM Journal of Research an Development, 22 pp. 393-404, April 1978.

[Willrich 91] WILLRICH R. *"Uma Proposta de um Modelo de Implementação de Serviços de Apoio a Aplicações Industriais Segundo o Padrão MMS"*, Dissertação de Mestrado, CPGEEL-UFSC, Florianópolis-SC, abril de 1991.

[Zafiropulo 80] ZAFIROPULO P., WEST C.H., COWAN D.D., BRAND D. *"Toward analysing and synthesizing protocols"*. IEEE Transactions on Communications, COM-28 pp. 651-660, Abril 1980.

[Zimmermann 80] ZIMMERMANN H. *"OSI Reference model : the ISO model of architecture for Open Systems Interconnection"*, IEEE Transactions on Communications, Vol. COM-28, n° 4, Abril 1980, pp. 425-432.

ANEXO I

ALGORITMO DO CAMINHO MAIS CURTO

Antes de se descrever o algoritmo algumas definições básicas devem ser dadas.

onde :

Pi(i) : tamanho do caminho mais curto para o estado i

k : número da interação

S : conjunto dos estados atuais

S_k : conjunto dos estados iniciais para a interação k

X : conjunto de todos os estados da FSM

$\overline{S} = X - S$

O algoritmo do caminho mais curto é constituído dos seguintes três passos :

1. Fazer :

Pi(estado inicial) = 0, Pi(i) = $+\infty$ para $i \geq$ estado inicial.

k = 0, S = {1}, $S_0 = \{1\}$

2. A cada interação K, seja $S_k = \{i \mid Pi(i) = k\}$ e $S = \{i \mid Pi(i) \leq k\}$.

Fazer

$S_{k+1} = \Gamma S_k \cap \overline{S}$

Pi(i) = k + 1 para $i \in S_{k+1}$

$S \leftarrow S \cup S_k$

3. Se |S| = |X|, FIM, senão, permanecer em 2 com $k \leftarrow k + 1$.

Um exemplo de aplicação deste algoritmo, para a FSM especificada na figura I.1, é encontrado abaixo :

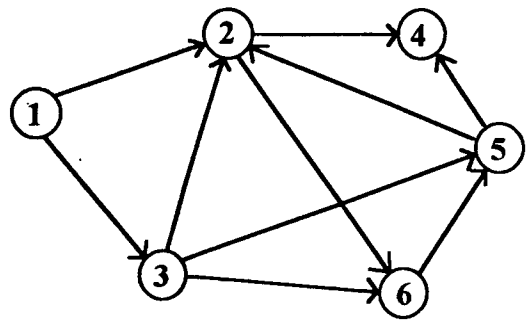

***Figura I.1** - Exemplo de aplicação para o algoritmo do caminho mais curto*

$X = \{1, 2, 3, 4, 5, 6\}$

1. $Pi(1) = 0, Pi(2) = Pi(3) = Pi(4) = Pi(5) = Pi(6) = +\infty, S = \{1\}$

2. $k = 0, S_0 = \{1\}, \Gamma S_0 \cap \overline{S} = \{2, 3\}, Pi(2) = Pi(3) = 1$

3. $S_1 = \{2, 3\}, S = \{1, 2, 3\}$

2. $k = 1, \Gamma S_1 \cap \overline{S} = \{4, 5, 6\}, Pi(4) = Pi(6) = Pi(5) = 2$

3. FIM

ANEXO II

ESPECIFICAÇÃO ESTELLE* PARA O PROTOCOLO TRANSPORTE SIMPLIFICADO

```
SPECIFICATION Transp_Simpl SYSTEMACTIVITY;
DEFAULT INDIVIDUAL QUEUE;

CHANNEL CH1 (user,t_ent);
 BY user  : TCONreq; TCONrsp; TDISreq; TDISrsp; TDATAreq; NULL;
 BY t_ent : TCONind; TCONconf; TDISind; TDISconf; TDATAind;

CHANNEL CH2 (t_ent,net);
 BY t_ent,net : CR; CC; DR; DC; DT; CC1; CC2;

MODULE User_head ACTIVITY;
IP TSAP : CH1(user)        NO QUEUE;
END; { User_A_head }

MODULE T_ENT_A_head ACTIVITY;
IP TSAP : CH1(t_ent)       NO QUEUE;
  NSAP : CH2(t_ent)        NO QUEUE;
END; { T_ENT_A_head }

MODULE NET_head ACTIVITY;
IP NSAP_A : CH2(net)       NO QUEUE;
END;

BODY User_body FOR User_head;

TRANS

TSAP!NULL
  begin  end;

TSAP!TCONreq
  begin  end;

TRANS
TSAP?TCONind
  begin  end;

TRANS
TSAP!TDATAreq
  begin  end;

TRANS
TSAP?TDATAind
  begin  end;
```

```
TRANS
TSAP!TDISreq
  begin end;

TSAP!TDISrsp
  begin  end;

TRANS
TSAP?TDISind
  begin  end;

TRANS
TSAP?TDISconf
  begin  end;

TRANS
TSAP!TCONrsp
  begin  end;

TRANS
TSAP?TCONconf
  begin  end;

END; { User_body }

BODY T_ENT_A_body FOR T_ENT_A_head;

STATE closed,wait_TCONrsp,wait_CC,closing,wait_DC,open,
    X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X16;

INITIALIZE TO closed
  begin  end;

TRANS
FROM closed TO X1
  TSAP?TCONreq
    begin end;

FROM closed TO X2
  NSAP?CR
    begin end;

FROM X1 TO wait_CC
  NSAP!CR
    begin  end;
```

```
FROM X2 TO wait_TCONrsp
  TSAP!TCONind
    begin end;

FROM wait_TCONrsp TO X3
  TSAP?TCONrsp
        begin end;

FROM wait_TCONrsp TO X4
  TSAP?TDISreq
        begin end;

FROM X3 TO open
  NSAP!CC
    begin  end;

FROM X4 TO closing
  NSAP!DR
        begin  end;

FROM closing TO X15
  TSAP?NULL
    begin  end;

FROM X15 TO closed
  TSAP!TDISconf
    begin  end;

FROM wait_CC TO X5
  NSAP?CC1
    begin end;

FROM X5 TO X6
  TSAP!TDISind
    begin  end;

FROM X6 TO wait_DC
  NSAP!DR
        begin  end;

FROM wait_DC TO X7
  NSAP?DC
        begin  end;

FROM X7 TO closed
  TSAP!TDISconf
        begin  end;
```

```
FROM wait_CC TO X8
  NSAP?CC2
        begin end;

FROM X8 TO open
  TSAP!TCONconf
        begin end;

FROM open TO X9
  TSAP?TDATAreq
   begin end;

FROM X9 to OPEN
  NSAP!DT
        begin end;

FROM open TO X10
  NSAP?DT
        begin end;

FROM X10 TO open
  TSAP!TDATAind
        begin end;

FROM open TO X16
  TSAP?NULL
     begin end;

FROM X16 TO X11
  TSAP!TDISind
        begin end;

FROM X11 TO wait_DC
  NSAP!DR
        begin end;

FROM open TO X12
  TSAP?TDISreq
        begin end;

FROM X12 TO wait_DC
  NSAP!DR
        begin end;

FROM open TO X13
  NSAP?DR
        begin end;

FROM X13 TO X14
```

```
  NSAP!DC
        begin  end;

FROM X14 TO closing
  TSAP!TDISind
        begin end;

END; { T_ENT_A_body }

BODY NET_body FOR NET_head;

TRANS
  NSAP_A?CR
        begin  end;

TRANS
  NSAP_A!CR
        begin  end;

TRANS
 NSAP_A?CC
        begin end;

TRANS
  NSAP_A!CC1
        begin end;

TRANS
  NSAP_A!CC2
        begin end;

TRANS
  NSAP_A?DR
        begin end;

TRANS
  NSAP_A!DR
        begin end;

TRANS
  NSAP_A?DC
        begin end;
```

```
TRANS
  NSAP_A!DC
        begin end;

TRANS
  NSAP_A?DT
        begin end;

TRANS
  NSAP_A!DT
        begin end;

END; { NET_body }

MODVAR
      User_A  : User_head;
      T_Ent_A : T_ENT_A_head;
      Network : NET_head;

INITIALIZE
  BEGIN
      INIT User_A WITH User_body;
      INIT T_Ent_A WITH T_ENT_A_body;
      INIT Network WITH NET_body;
      CONNECT User_A.TSAP TO T_Ent_A.TSAP;
      CONNECT T_Ent_A.NSAP TO Network.NSAP_A;
    END;
END. { SPECIFICATION Transp_Simpl }
```

ANEXO III

EXEMPLOS DE ARQUIVOS

A figura abaixo mostra um exemplo de um arquivo gerado pela ferramenta ESTIM, extensão *.proj.lab.l.auto* :

```
last_state_ref(18).
last_lab_ref(25).
arc(1,[T_ENT_A(input(NSAP(CR)))],[T_ENT_A(T2),NETWORK(T2)],3).
arc(1,[T_ENT_A(input(TSAP(TCONREQ)))],[T_ENT_A(T1),USER_A(T2)],2).
arc(2,[T_ENT_A(output(NSAP(CR)))],[NETWORK(T1),T_ENT_A(T3)],4).
arc(3,[T_ENT_A(output(TSAP(TCONIND)))],[USER_A(T3),T_ENT_A(T4)],5).
arc(4,[T_ENT_A(input(NSAP(CC1)))],[T_ENT_A(T11),NETWORK(T4)],6).
arc(4,[T_ENT_A(input(NSAP(CC2)))],[T_ENT_A(T16),NETWORK(T5)],8).
arc(5,[T_ENT_A(input(TSAP(TCONRSP)))],[T_ENT_A(T5),USER_A(T10)],9).
arc(5,[T_ENT_A(input(TSAP(TDISREQ)))],[T_ENT_A(T6),USER_A(T6)],10).
arc(6,[T_ENT_A(output(TSAP(TDISIND)))],[USER_A(T8),T_ENT_A(T23)],11).
arc(7,[T_ENT_A(output(TSAP(TDISIND)))],[USER_A(T8),T_ENT_A(T29)],13).
arc(8,[T_ENT_A(output(TSAP(TCONCONF)))],[USER_A(T11),T_ENT_A(T17)],12).
arc(9,[T_ENT_A(output(NSAP(CC)))],[NETWORK(T3),T_ENT_A(T7)],12).
arc(10,[T_ENT_A(output(NSAP(DR)))],[NETWORK(T6),T_ENT_A(T8)],13).
arc(11,[T_ENT_A(output(NSAP(DR)))],[NETWORK(T6),T_ENT_A(T24)],14).
arc(12,[T_ENT_A(input(NSAP(DR)))],[T_ENT_A(T27),NETWORK(T7)],17).
arc(12,[T_ENT_A(input(NSAP(DT)))],[T_ENT_A(T20),NETWORK(T11)],16).
arc(12,[T_ENT_A(input(TSAP(NULL)))],[T_ENT_A(T22),USER_A(T1)],6).
arc(12,[T_ENT_A(input(TSAP(TDATAREQ)))],[T_ENT_A(T18),USER_A(T4)],15).
arc(12,[T_ENT_A(input(TSAP(TDISREQ)))],[T_ENT_A(T25),USER_A(T6)],11).
arc(13,[T_ENT_A(input(TSAP(NULL)))],[T_ENT_A(T9),USER_A(T1)],18).
arc(14,[T_ENT_A(input(NSAP(DC)))],[T_ENT_A(T14),NETWORK(T9)],18).
arc(15,[T_ENT_A(output(NSAP(DT)))],[NETWORK(T10),T_ENT_A(T19)],12).
arc(16,[T_ENT_A(output(TSAP(TDATAIND)))],[USER_A(T5),T_ENT_A(T21)],12).
arc(17,[T_ENT_A(output(NSAP(DC)))],[NETWORK(T8),T_ENT_A(T28)],7).
arc(18,[T_ENT_A(output(TSAP(TDISCONF)))],[USER_A(T9),T_ENT_A(T15)],1).
$
```

***Figura** - Exemplo de um arquivo gerado pelo ESTIM*

A figura abaixo mostra um exemplo de um arquivo gerado após a primeira transformação, extensão *.tim* :

```
(1, CR, nil, 3)
(1, TCONREQ, nil, 2)
(2, nil, CR, 4)
(3, nil, TCONIND, 5)
(4, CC1, nil, 6)
(4, CC2, nil, 8)
(5, TCONRSP, nil, 9)
(5, TDISREQ, nil, 10)
(6, nil, TDISIND, 11)
(7, nil, TDISIND, 13)
(8, nil, TCONCONF, 12)
(9, nil, CC, 12)
(10, nil, DR, 13)
(11, nil, DR, 14)
(12, DR, nil, 17)
(12, DT, nil, 16)
(12, NULL, nil, 6)
(12, TDATAREQ, nil, 15)
(12, TDISREQ, nil, 11)
(13, NULL, nil, 18)
(14, DC, nil, 18)
(15, nil, DT, 12)
(16, nil, TDATAIND, 12)
(17, nil, DC, 7)
(18, nil, TDISCONF, 1)
```

Figura** - Exemplo de um arquivo gerado pela ferramenta **transf